Sistemas Informáticos
Curso 2005-06

Sistema multiusuario para la gestión universitaria vía Web, incluyendo el desarrollo de un interfaz para el diseño de aplicaciones de datos visuales

Roberto Carrasco Sánchez
José Javier Crespo Ábalos
Antonio José López Mancheño

Dirigido por:
Isabel Pita Andreu
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

Resumen en español.

El objetivo del proyecto es la enseñanza a distancia de tipos abstractos de datos (TAD's), así como aplicaciones de los mismos. Se hace un recorrido por los principales tipos de datos, desde los mas sencillos (pilas, colas...) hasta otros mas complicados (como aplicaciones de los grafos). El medio para la enseñanza de los mismos es el de animaciones y películas ricas en contenido visual, que permiten su visualización a través de cualquier navegador (dotado con el plugin Flash Macromedia). El alumno tiene control sobre las animaciones, permitiendo su ejecución paso a paso, de forma continuada, repitiendo determinadas partes, etc. En todos los casos la dificultad de las mismas dentro de cada tipo de datos es progresiva, permitiendo una paulatina mejora de los conocimientos.

Todas las animaciones se integran en un entorno Web. También se utiliza un servidor de aplicaciones y una base de datos para dar soporte a este entorno. De esta forma se permite su instalación en un servidor para que los clientes (los alumnos en este caso) se conecten al mismo sea cual sea su ubicación y puedan acceder al entorno, previa verificación de su identidad en la base de datos de usuarios.

Resumen en inglés.

The main purpose of this project is to help students on learning abstract data types and their applications. We start by defining the basic abstract data types: stack, queue, etc. Then, we define more complex abstract data types such as heaps, red-black trees and graph applications. The learning methodology is based in animations and movies with a lot of visual information which allow the student to visualize the abstract data types using any navigator tool that has plug-in Flash macromedia. The student has some control over the animations. He/she can execute them step by step, and go forward and backward in order to repeat some parts. In any case, the difficulty within each data types will increase, so the student can improve his/her knowledge slowly.

All the animations are made up in a web environment. Also we use a server and a database to support this environment. In this way we can install it in a server for the customer (in this case the students). The students can connect with this server in any location and access to this environment. In this case the users need to verify his/her identity in the user database.

1.- Introducción

El objeto de este proyecto es diseñar una serie de animaciones que faciliten a los alumnos de cualquier titulación de informática aprender por sí mismo tipos de abstractos de datos (TAD).

Las animaciones que se han desarrollado han sido elegidas intentando cubrir el mayor espectro posible de las estructuras de datos que se estudian en asignaturas de estructuras de datos de cualquier Facultad de Informática. Así, en este proyecto se puede ver un amplio recorrido de distintos tipos de datos, desde el funcionamiento de tipos de datos simples (como las pilas o las colas) hasta problemas complejos sobre grafos (cálculo del árbol de recubrimiento mínimo por el algoritmo de Prim o el algoritmo de Dijkstra para el cálculo de caminos mínimos) pasando por la estructura de distintos tipos de árboles binarios.

Se distinguen dos tipos de animaciones dependiendo de la información que le muestran al alumno, una parte de ellas exponen al alumno un nuevo tipo abstracto de datos con la intención de explicarle el funcionamiento del mismo, y en el resto, se supone que el alumno conoce el TAD (o varios TAD distintos) y a través de un problema concreto se le enseña a usarlos o combinarlos para solucionar el problema en cuestión.

En función de la complejidad del concepto que se esté intentando transmitir las animaciones presentan características distintas, es decir, las animaciones que intentan enseñar conceptos básicos presentan sobre todo una carga visual pero no se enfatiza demasiado sobre los aspectos teóricos, sin embargo, en aquellas en las que se hace un seguimiento exhaustivo de un algoritmo más complejo o se soluciona un problema específico, no se le da tanta importancia a la carga visual como a la didáctica, en estas animaciones se muestra en cada momento cuál es la instrucción actual del algoritmo y los efectos que provoca en el problema que se intenta resolver.

Las animaciones de este proyecto se han diseñado en todo momento con el objeto de cubrir tres aspectos distintos:

- Ser fáciles de comprender por el alumno.
- Presentar las características necesarias para ser expuestas en público.
- Dotarlas de una gran carga visual para poder ser proyectadas en aulas de grandes dimensiones.

En todo momento se ha intentado cubrir todos los objetivos y generar un entorno cómodo tanto para el alumno como para el profesor que las exponga.

Las animaciones se han desarrollado mediante Macromedia Flash y todas ellas podrán ser visualizadas mediante cualquier navegador que disponga del plugin Macromedia Flash Player (descargable automáticamente por el navegador en caso de no poseerlo o manualmente a través de "`www.macromedia.com`").

Todas las animaciones se mostrarán a través de un entorno de páginas Web alojadas en un servidor de aplicaciones configurado para el proyecto. Asimismo, existe una base de datos (en principio tan solo para información de registro de usuarios) de forma que solo los visitantes que acrediten su identidad tendrán acceso al contenido. Además dicho entorno permite la posterior ampliación organizada de contenidos y servicios.

2.- Animaciones

En función de la ayuda que le suponen al alumno y de lo que se desarrolla en ellas, podemos distinguir tres tipos de animaciones:

- Complementarias: se muestra la definición de un tipo abstracto de datos (TAD).
  El contenido de la animación puede ser comprendido sin excesiva dificultad por parte del alumno, y la animación ayuda al alumno o bien a comprender rápidamente el tipo de datos que se esté tratando (si no ha recibido ninguna explicación previa) o bien a afianzar el entendimiento de la estructura (gracias a que puede ver una animación dinámica en la que se le muestra visualmente el concepto) si es que ya había recibido alguna explicación.
  Dentro de este grupo se encuentran las animaciones: pilas (2.1.1), colas (2.2.1) y árboles binarios de búsqueda (2.3.1).

- Seguimiento completo: se muestra instrucción por instrucción el desarrollo completo de un algoritmo de dificultad entre media y alta, desde el principio hasta el final.
  El alumno encontrará algoritmos en los que se muestra la inserción y la eliminación para un TAD determinado o bien se tratará de un algoritmo que se ayuda de ese TAD para solucionar un problema de la vida real o académico.
  El objeto de estas animaciones es enseñar al alumno el funcionamiento de un algoritmo complejo paso a paso.
  El desarrollo de animaciones de este tipo puede ser especialmente importante en el aprendizaje de un alumno de cualquier titulación informática, porque ayuda a comprender un algoritmo complejo en poco tiempo. El esfuerzo que debe hacer el alumno para comprender el algoritmo se verá reducido si puede ir viendo paso a paso como avanza el algoritmo, es decir, si no necesita ir imaginando en su cabeza como avanza el algoritmo porque ya se lo están mostrando.

  Se ha hecho un seguimiento completo a algoritmos de: pilas (2.1.2 y 2.1.3), colas (2.2.2), árboles binarios (2.3.2) y grafos (2.5.1, 2.5.2 y 2.5.3.).

- Mixtas: se trata de una mezcla de los tipos anteriores, en ellas, en primer lugar, se define un tipo abstracto de datos y después se muestra el funcionamiento del mismo.
  El objeto de estas animaciones es el mismo que las del tipo anterior.

Los TADS así desarrollados son: árboles rojinegros (2.3.3) y montículos (2.4.1)

Las animaciones se han diseñado intentando cumplir dos objetivos: poder ser expuestas por el profesor en clase (por medio de un proyector) y poder ser estudiadas por cada alumno de forma individual. El primer motivo es la causa de que las animaciones presenten colores vivos y estructuras de datos con un tamaño notable, y es que si las animaciones son expuestas en un aula de gran tamaño los alumnos de las últimas filas no podrían seguir la animación. Para que las dos perspectivas pudieran ser complementarias se añadió el sistema de navegación en forma de botones que presentan las animaciones. Hay dos sistemas de navegación: con dos o con cinco botones. Las animaciones que constan de dos botones (bobinar y rebobinar) son las más carentes de movimiento pero muy cargadas de contenido teórico, y las de cinco botones (ejecutar, parar, pausa, bobinar y rebobinar) son las que presentan más movimiento (y no necesariamente una carencia de contenido teórico). Nótese que en las animaciones con sólo dos botones las imágenes se encuentran detenidas y podemos avanzar de la vista actual, a la anterior, o a la siguiente mediante los botones. En el caso de las animaciones con cinco botones se avanzará mediante el botón ejecutar si lo que queremos es continuar a la vista siguiente, o mediante bobinar si no queremos ver terminar la vista actual porque preferimos ir a la siguiente vista. En estas animaciones con el botón rebobinar volvemos a la vista anterior, con el botón pausa detenemos la animación y con el botón parar regresamos al inicio de la animación.

2.1.- Pilas

El objetivo del proyecto es crear un entorno Web de aprendizaje de estructuras de datos que haga un recorrido amplio por las distintas estructuras de datos, desde las más simples hasta las más complejas. Este objetivo no se habría cumplido con solvencia si no hubiéramos incluido las pilas en el proyecto, ya que las pilas es el primer TAD que se estudia en cualquier asignatura de estructuras de datos. Por este motivo decidimos incorporar la animación “El TAD de las pilas” (2.1.1), que muestra el funcionamiento de una pila.

Una pila es una estructura de datos en la cual el acceso está limitado al elemento más recientemente insertado, es decir, sólo se puede insertar “sobre” el último elemento insertado y sólo se puede sacar el último elemento insertado, no tenemos la posibilidad de acceder a ningún elemento intermedio. Se comporta de forma muy similar a una pila de platos de nuestra vida cotidiana, si apilamos todos los platos de un banquete uno encima de otro sólo podríamos añadir uno encima del último y sólo podríamos obtener el último de ellos, porque cualquier intento de acceder a uno intermedio terminaría con toda la pila derramada.

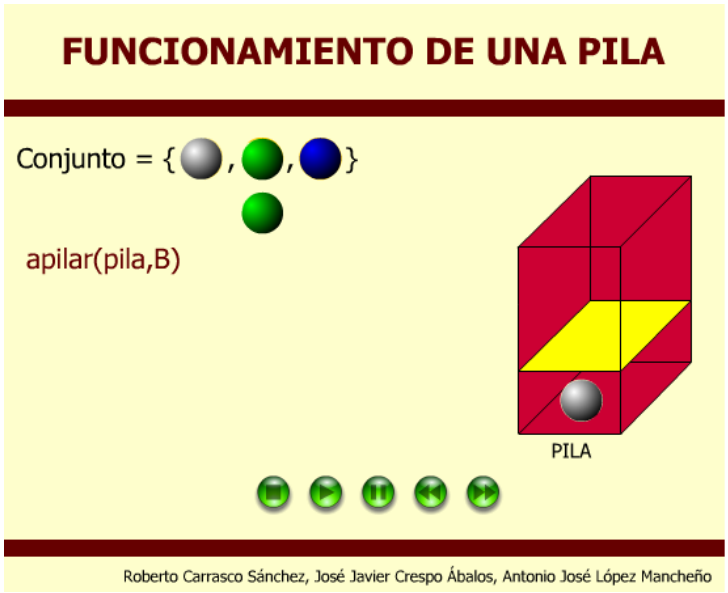
Este tipo de ejemplos cotidianos en los que usamos una pila fue lo que dio lugar al desarrollo de la animación “Uso en una cinta corredera” (2.1.3). Las pilas tienen un papel fundamental en gran variedad de algoritmos, un ejemplo de ello es la animación “Transformación de expresiones infijas a postfijas” (2.1.2). Aunque se pueden emplear otras estructuras de datos para resolver el problema (colas dobles, secuencias…), se utilizan pilas porque dan lugar a un algoritmo muy sencillo y eficiente.

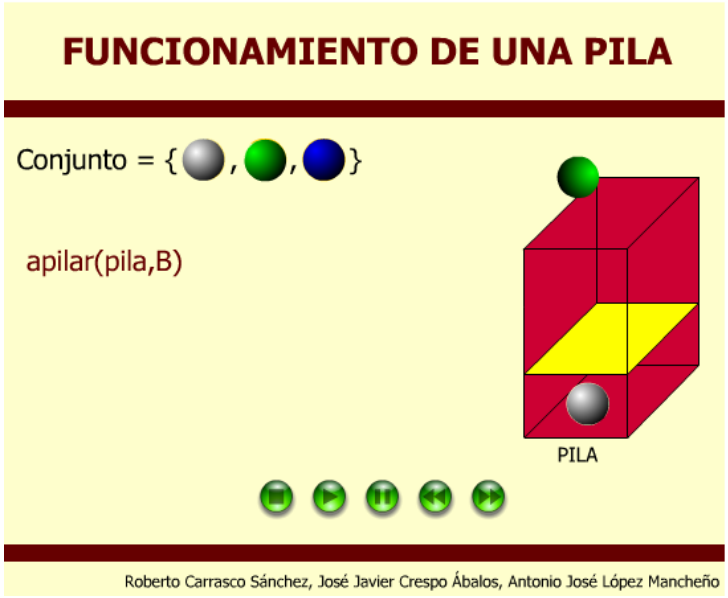
Una vez que el lector conoce el motivo de la inclusión de cada una de las animaciones vamos a pasar a comentar los aspectos relevantes de las mismas detalladamente.

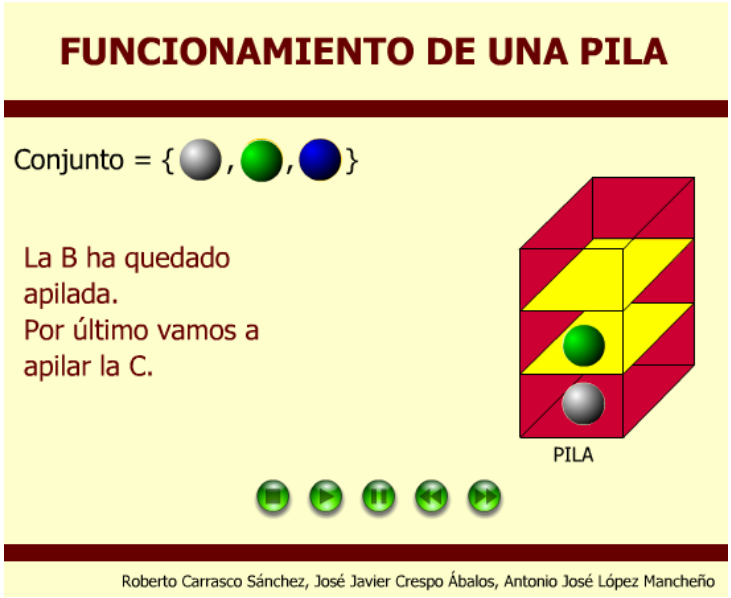
2.1.1 El TAD de las pilas

Se trata de una animación de carácter complementario en la que se muestra el funcionamiento de una pila. La animación consta de cinco botones porque presenta mucho movimiento. En ella se le enseña al alumno en qué consisten las operaciones de las pilas: apilar, desapilar, y consultar la cima. La animación empieza identificándole al alumno qué es el TAD pila y cuál es el conjunto de elementos que se van a apilar, después se va apilando uno a uno hasta que la pila queda completa, a continuación se muestra qué elemento es la cima de la pila y empiezan a desapilarse todos hasta que la pila vuelve a quedar vacía.

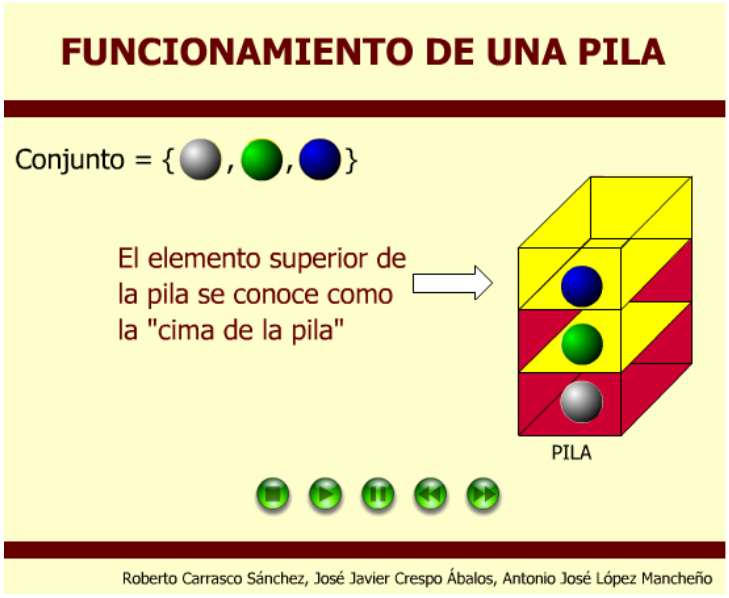
Apilar:



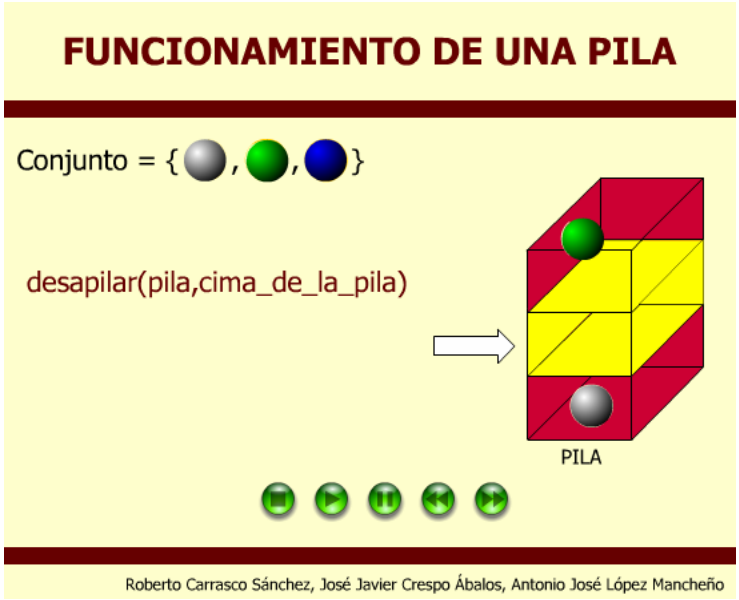

FUNCIONAMIENTO DE UNA PILA
Conjunto = { , , }
apilar(pila,B)
PILA
Roberto Carrasco Sánchez, José Javier Crespo Ábalos, Antonio José López Mancheño


FUNCIONAMIENTO DE UNA PILA
Conjunto = { , , }
La B ha quedado apilada.
Por último vamos a apilar la C.
PILA
Roberto Carrasco Sánchez, José Javier Crespo Ábalos, Antonio José López Mancheño

Cima de la pila:



Desapilar:

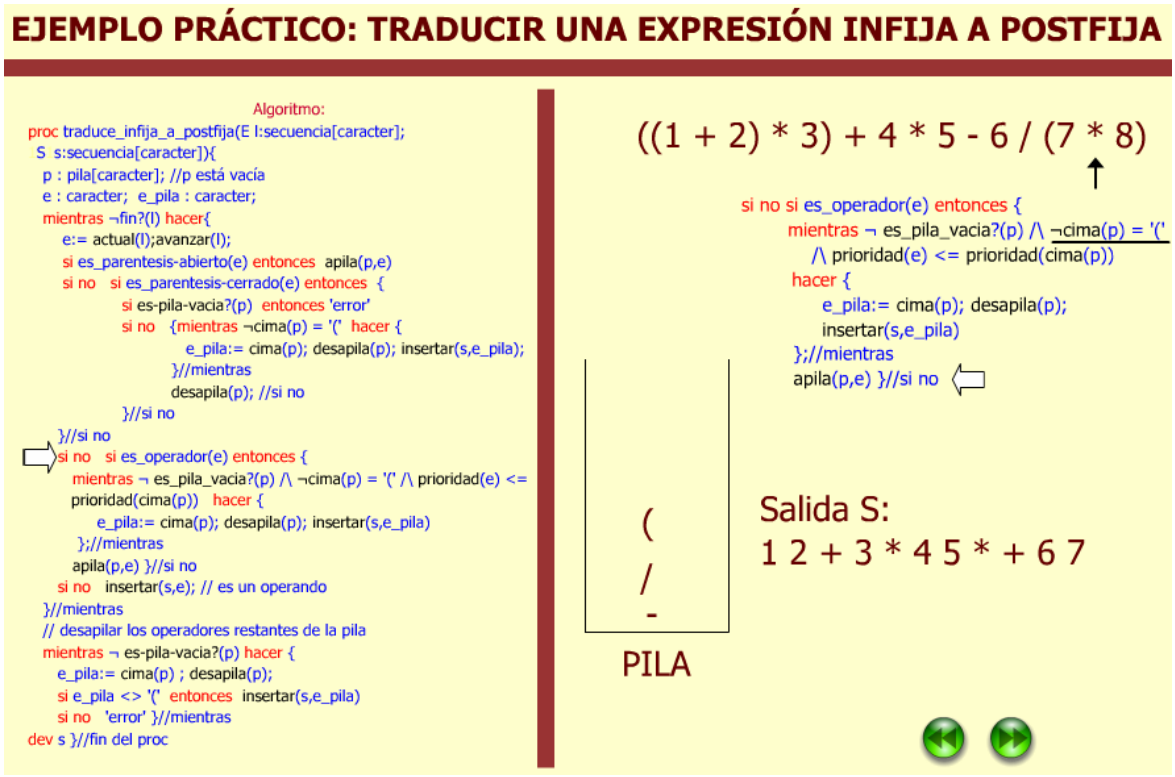


2.1.2 Transformación de expresiones infijas a postfijas

Es una animación de seguimiento completo en la que se muestra paso a paso como podemos transformar mediante un algoritmo una expresión aritmética expresada en forma infija a postfija*.

La animación está estructurada de tal modo que el usuario encuentra en la parte izquierda de la misma el algoritmo completo junto con una flecha que muestra en cada momento en qué parte del algoritmo se encuentra la animación. Y a la derecha puede verse un zoom de la zona del algoritmo que se está atravesando, acompañada por una flecha que muestra qué línea de código se está ejecutando en el momento actual. También puede verse la pila (a la que se añadirán o eliminarán elementos según lo demande el algoritmo) y la secuencia de salida. Se muestra el contenido de cada una de estas estructuras según progresa el algoritmo.

La animación muestra como se resuelve el problema paso a paso. En todos los pasos se explica porque se toman las decisiones, en base al valor de las expresiones condicionales de las instrucciones condicionales e iterativas.

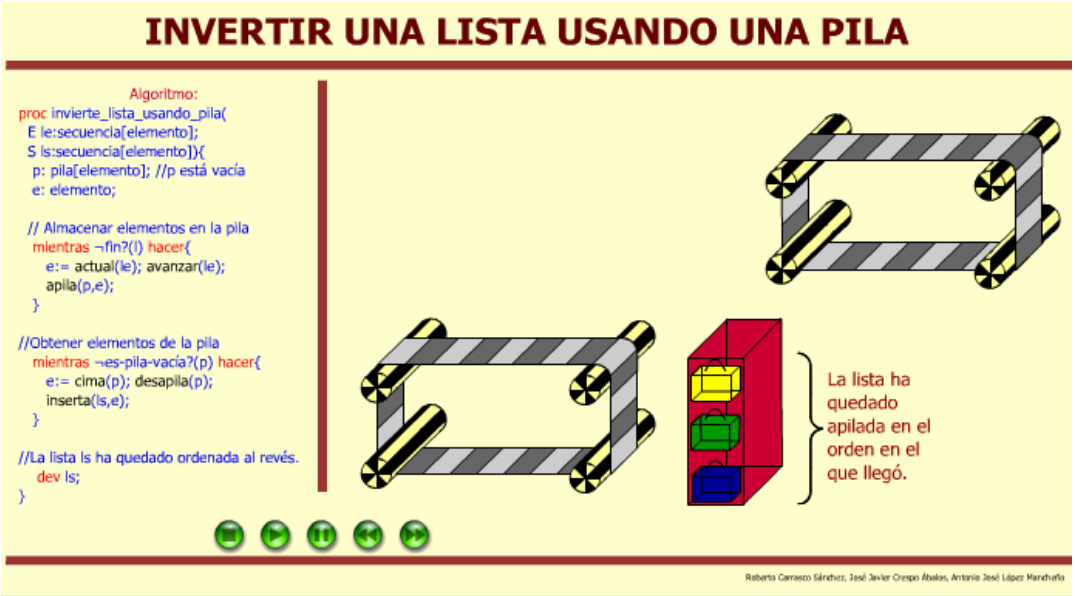


* Las expresiones aritméticas pueden expresarse en tres formas distintas en función de cómo estén situados sus operandos y sus operadores. La forma infija es aquella en la que todos hemos aprendido matemáticas, es decir, los operadores se encuentran entre los operando y la prioridad de un operador u otro de denota mediante el uso de paréntesis (por ejemplo: (b+c) * a). En una expresión postfija los operadores se sitúan al después de los operandos (siguiendo con el ejemplo anterior: b c + a *). Y por último las expresiones prefijas son aquellas en las que el operador se sitúan antes que los operandos (en el ejemplo correspondiente: * a + b c).

2.1.3 Uso en una cinta corredera

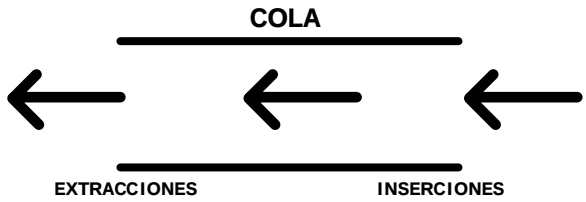
En este caso tenemos una animación de carácter mixto. En esta animación se intenta conseguir que el alumno perciba como algunos TAD´s que estudia están presentes en nuestras actividades diarias. Así, se ilustra un caso en el que hay una serie de maletas deslizándose a lo largo de una cinta corredera (emulan ser una lista), al llegar al final de la cinta las maletas caen una detrás de la otra, y todas ellas van a parar a una estructura (una pila) en la que se amontonan unas encima de las otras. Después se desapilan una detrás de otra y acceden a otra cinta por la que continuarán desplazándose, pero en este caso en orden inverso al inicial.

Con este ejemplo conseguimos transmitir al alumno como una lista puede invertir el orden de sus elementos utilizando una pila. Todo ello va acompañado del correspondiente seguimiento del algoritmo, en el que se hace corresponder cada circunstancia de la animación con la orden adecuada del mismo, por ejemplo, cuando las maletas caen a la pila se ilumina la instrucción apilar, o cuando las maletas vuelven a formar parte de la cinta se ilumina la orden insertar.



2.2.- Colas

Una Cola es una estructura de datos de tipo FIFO (First-In First-Out o "el primero que entra es el primero que sale") en este tipo de datos las inserciones se realizan por un extremo y las extracciones por el otro; el único elemento que se puede consultar en todo momento es el que primero se insertó.

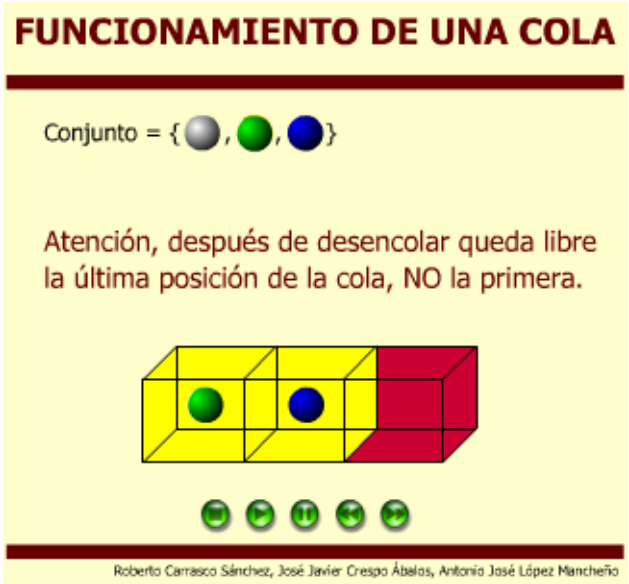


Este tipo de datos lineal siempre se explica en todas las asignaturas de estructuras de datos a continuación de las pilas y era especialmente importante mostrar su comportamiento en este proyecto porque es un TAD con gran cantidad de aplicaciones y apariciones en cualquier titulación de informática: desde las colas de impresión (informática general), asignación del procesador a procesos de usuario por el sistema operativo (políticas como round-robin).

Para este TAD hemos generado dos animaciones, una introductoria al concepto de cola y otra en la que se usa en un problema específico.

2.2.1 El TAD de las colas

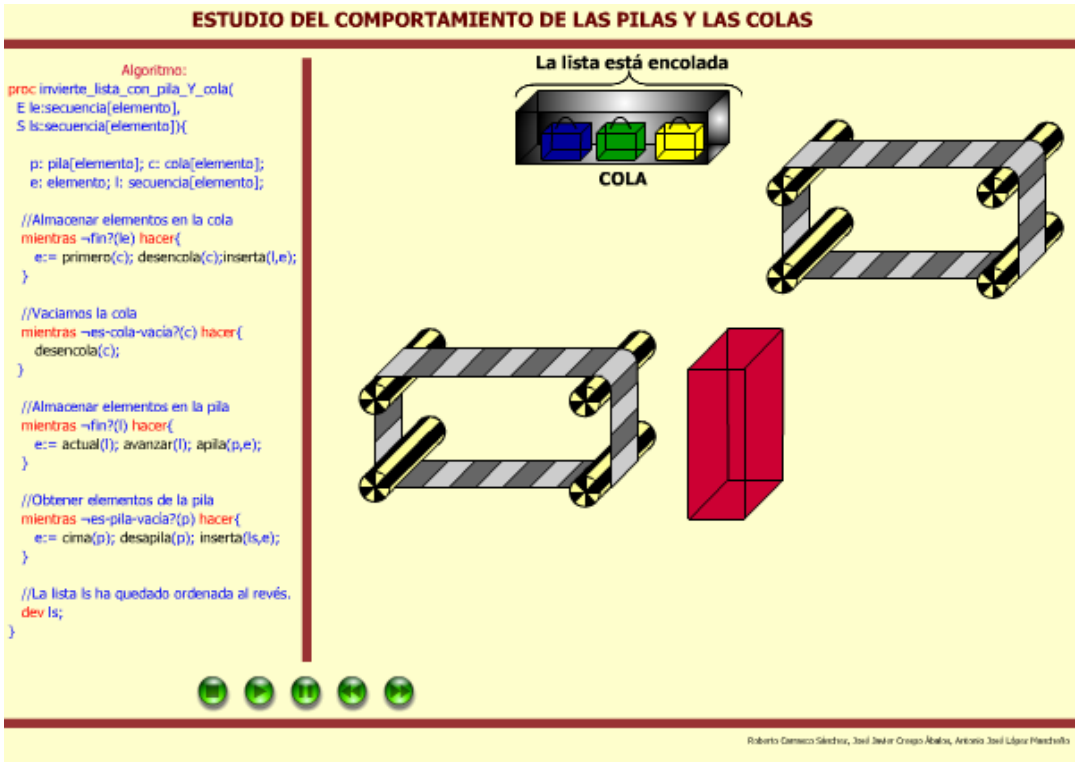
Se trata de una animación de carácter complementario, en la que se enseña a los alumnos el concepto de cola y las funciones básicas que pueden aplicarse sobre la misma. Todos los elementos, tanto la cola como las esferas que se insertan están dibujados en tres dimensiones para intentar hacerlo más espectacular, incrementar la carga visual.



Inicialmente se muestra al alumno el sentido de la inserción y de la extracción, y posteriormente se van insertando las esferas hasta que la cola se encuentra llena. Después de eso, se extraen todos los elementos hasta que la cola vuelve a quedar vacía.

2.2.2 Uso en una cinta corredera con cola intermedia

Es una animación de seguimiento completo. Es una ampliación del ejemplo 2.1.3 (uso de una cinta corredera) sólo que en este caso se añade una cola a la salida de la primera cinta, de tal modo, que todas las maletas se insertan en la cola antes de apilarse, y a partir de ahí el comportamiento es el mismo que en el ejemplo 2.1.3. Con esta animación intentamos afianzar los conceptos de pila y cola a los alumnos, para que sean conscientes de las distintas funcionalidades que presenta cada una y de en qué caso y con qué motivos es necesario usar un TAD o el otro.



2.3.- Árboles

En nuestro empeño de intentar abarcar la mayor parte posible del temario de una asignatura de estructuras de datos el siguiente paso es el de estudiar los árboles (binarios en nuestro caso).

Las estructuras arbóreas que estudiamos generalizan las estructuras lineales vistas anteriormente (pilas y colas), de forma que en vez de pasar de un dato al (único) siguiente, tenemos varios "siguientes" que se consideran los hijos del dato en cuestión.

Nosotros nos hemos centrado en los árboles binarios (son los más estudiados en las titulaciones de informática) que son aquellos en los que todo nodo tiene dos hijos a lo sumo (uno o ambos pueden ser vacíos). Y dentro de los árboles binarios hemos estudiado un caso particular de los mismos, los árboles binarios de búsqueda (ejemplo 2.3.1).

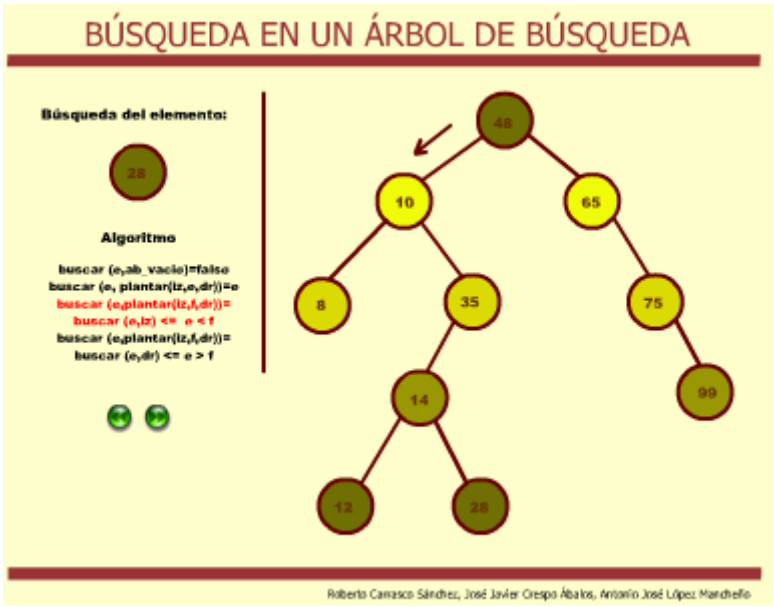
Los árboles binarios de búsqueda se caracterizan por presentar en sus nodos elementos ordenables y además siempre ocurre que el elemento de la raíz es mayor que todos los elementos de su subárbol izquierdo (el árbol que cuelga por la izquierda de ese nodo en particular) y menor que todos los elementos de su subárbol derecho (lo mismo que antes pero en este caso para el hijo derecho), y además los subárboles izquierdo y derecho son también árboles binarios de búsqueda.

Así las cosas, se decidió diseñar animaciones que explicaran la inserción, búsqueda y eliminación de elementos en un árbol binario de búsqueda, así como el recorrido en anchura de un árbol binario (por ser el recorrido que mas trabajo cuesta comprender a los alumnos) y se estudió un tipo particular de árboles binarios de búsqueda que son los árboles rojinegros.

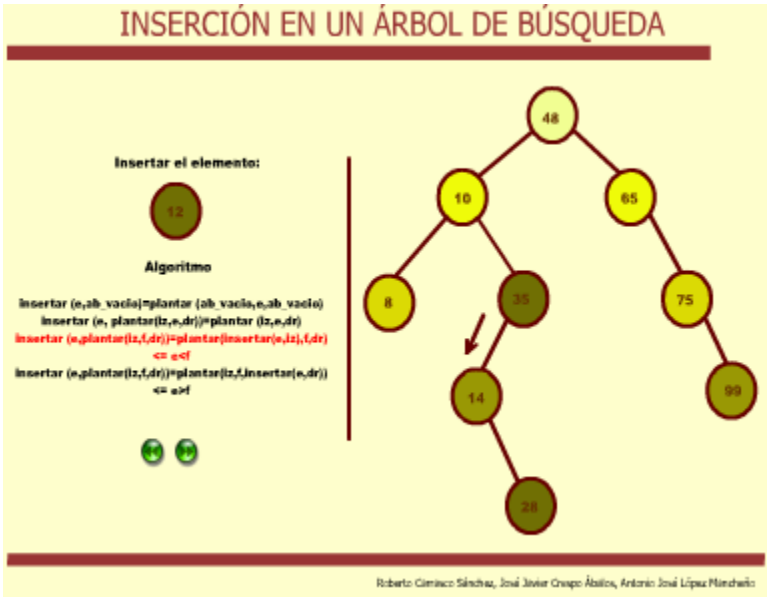
2.3.1 El TAD de los árboles binarios de búsqueda

Se han diseñado tres animaciones distintas, en ellas se ilustra la búsqueda, eliminación e inserción en un árbol binario de búsqueda. Todas las animaciones pueden considerarse mixtas, y nótese que la explicación va acompañada de las ecuaciones algebraicas que definen la instrucción que se está ejecutando. Con esto intentamos conseguir que el alumno identifique cada ecuación con una acción y comprenda por lo tanto el objeto de las ecuaciones algebraicas que definen el comportamiento de un TAD.

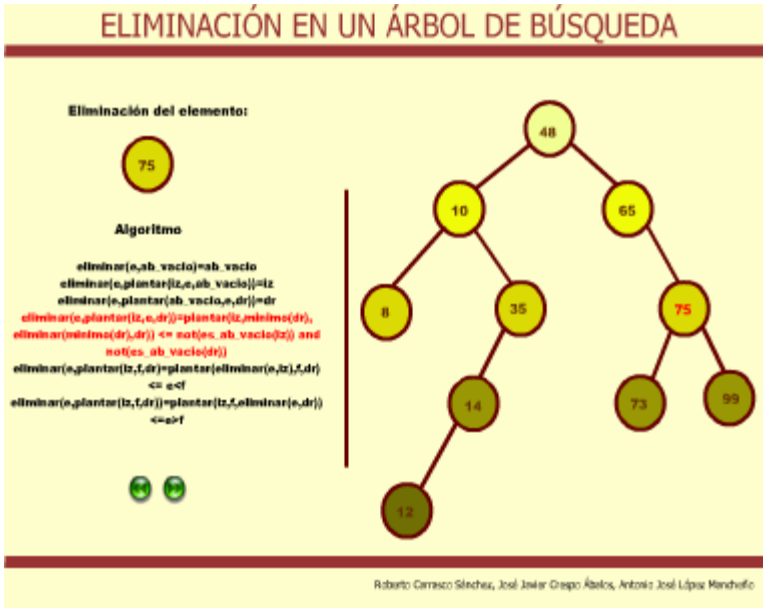
Búqueda:



Inserción:



Eliminación:



2.3.2 Recorrido en anchura de un árbol binario

Esta es una animación de seguimiento completo.

En la asignatura de Estructuras de datos se estudian dos tipos de recorrido: recorrido en profundidad y recorrido en anchura o a nivel. Puesto que los árboles no son secuenciales como las listas, hay que buscar estrategias alternativas para visitar todos los nodos.

El recorrido en anchura consiste en ir visitando el árbol por niveles. Primero se visitan los nodos de nivel 1 (como mucho hay uno, la raíz), después los nodos de nivel 2, así hasta que ya no queden más niveles.

El recorrido en anchura se implementa de forma iterativa, utilizando una cola como estructura de datos auxiliar. El procedimiento consiste en encolar (si no están vacíos) los subárboles izquierdo y derecho del nodo extraído de la cola, y seguir desencolando y encolando hasta que la cola esté vacía.

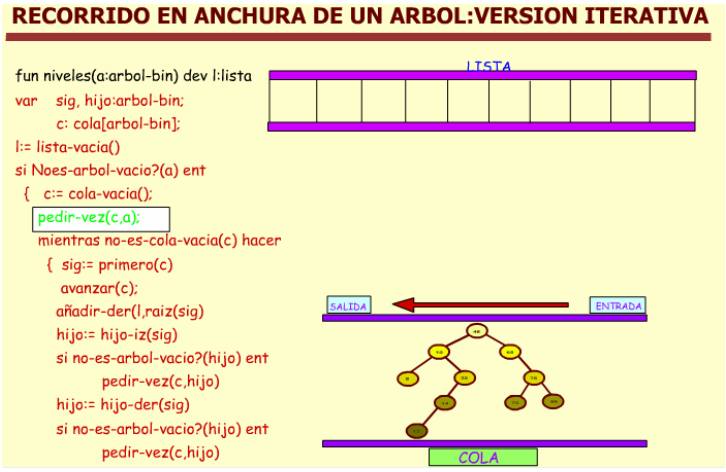
Implementación

```
Fun niveles(a:arbol-bin) dev lista l:lista
Var
        Sig, hijo: arbol-bin;
        C: cola[arbol-bin]
{
        l:=lista_vacia();
        si no_es_arbol_vacio?(a) entonces{
                C:= cola_vacia();
                Pedir_vez(c,a);
                mientras no_es_cola_vacia(c) hacer{
                        Sig:= primero(c);
                        Avanzar(c);
                        Añadir_der(l,raiz(sig));
                        Hijo:= hijo_izq(sig);
                        Si no_es_arbol_vacio(hijo) entonces{
                                Pedir_vez(c,hijo);
                        }
                        Hijo:= hijo_der(sig);
                        Si no_es_arbol_vacio(hijo) entonces{
                                Pedir_vez(c,hijo);
                        }
}}}
```

Procedimiento del algoritmo:

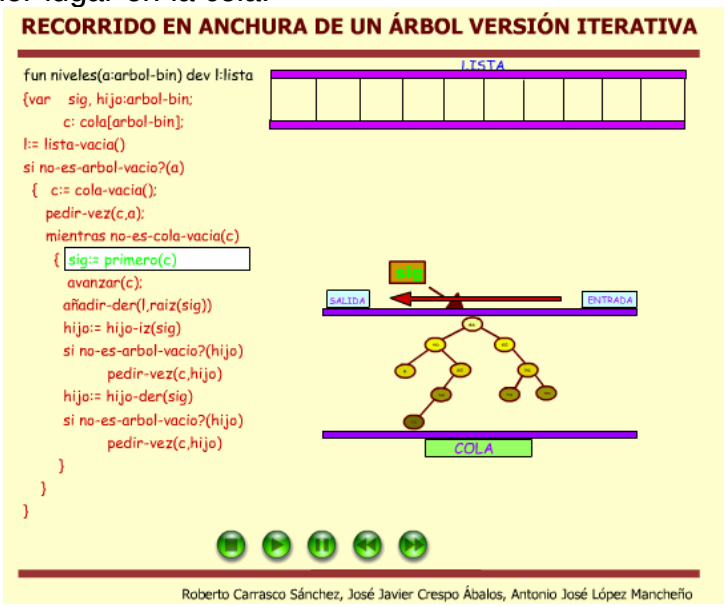
La implementación está realizada de forma iterativa, y utilizaremos para ello una cola para ir guardando los distintos subárboles que aparezcan, dos punteros a árboles para tener las direcciones de los distintos árboles y una lista para ir guardando el recorrido que vaya surgiendo.

En el momento inicial, introducimos todo el árbol en la cola.

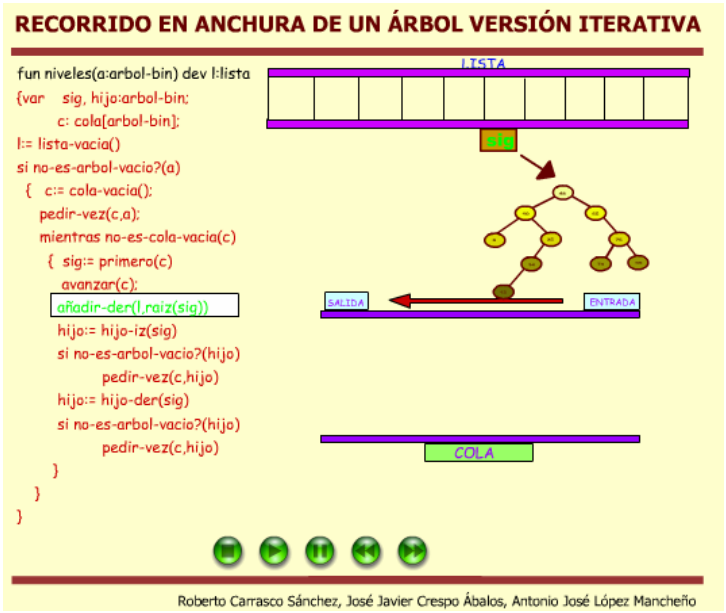


El procedimiento consiste en un bucle while, que se ejecutará mientras la cola no este vacía.

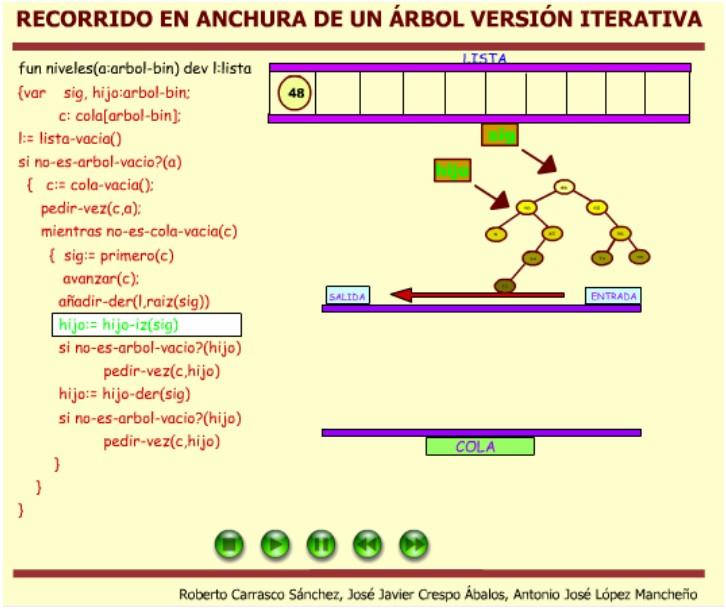
Guardaremos en un puntero(sig) la dirección de memoria del árbol que esté en primer lugar en la cola.



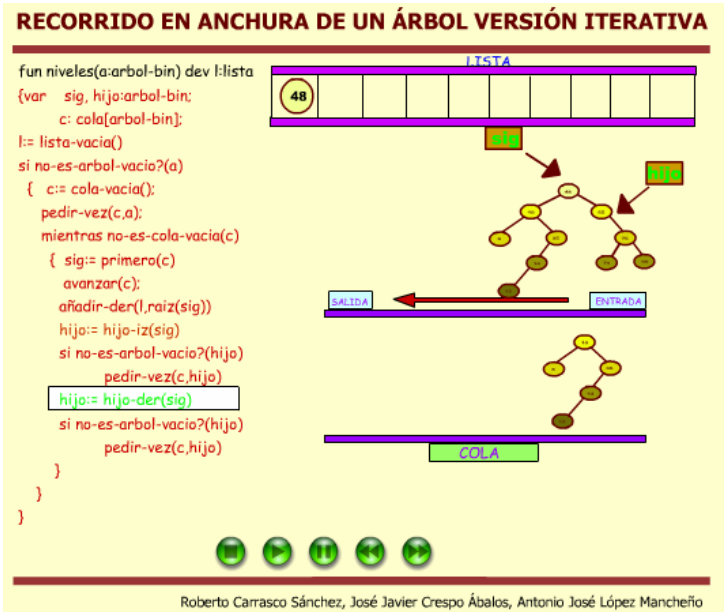
Avanzaremos la cola para sacar el árbol que esta en primer lugar.



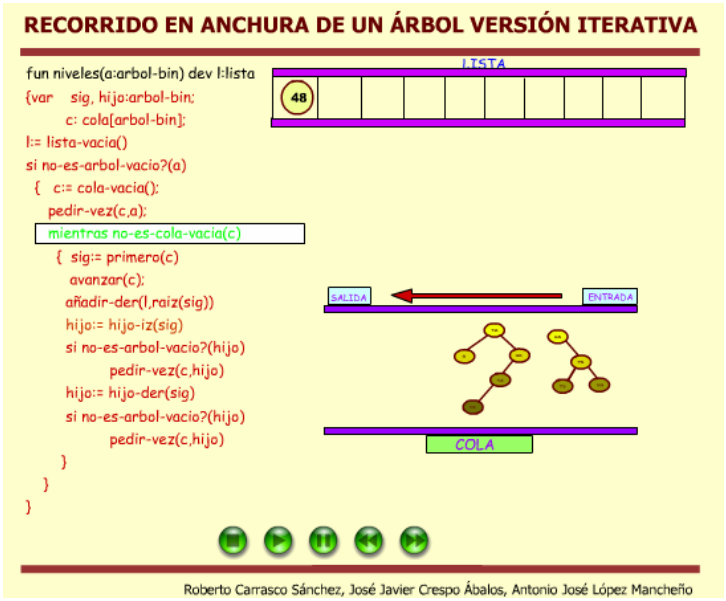
Añadiremos a la lista auxiliar que lleva el recorrido la raíz del árbol que tendíamos apuntado por el puntero sig.



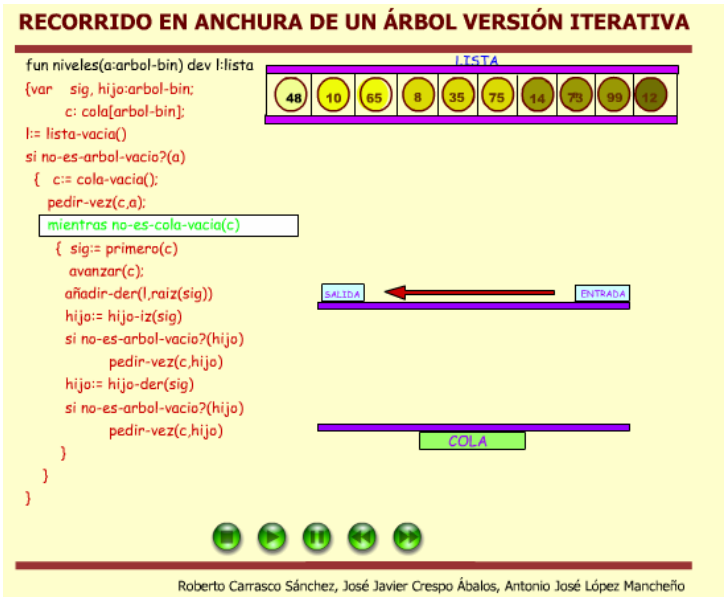
A continuación guardamos en el puntero hijo, la dirección de memoria del subárbol izquierdo del árbol apuntado por sig, y si no es vacío, lo introducimos en la cola.



Guardamos en otro puntero(hijo), la dirección de memoria del subárbol derecho del árbol apuntado por sig, y si no es vacío, lo introducimos en la cola.
Y volvemos a iterar estas instrucciones hasta que la cola se quede vacía.



Una vez que la cola está vacía, tenemos en la lista el recorrido en anchura del árbol analizado.



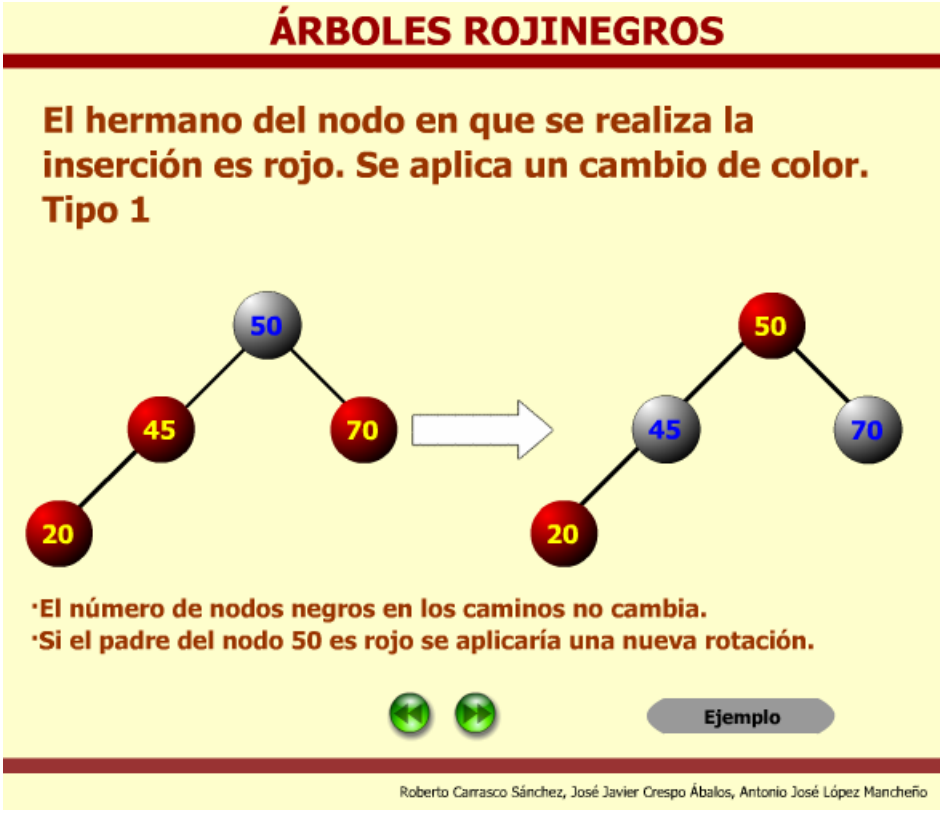
2.3.3 Árboles rojinegros

Los árboles rojinegros son un caso particular de árboles binarios de búsqueda. Se caracterizan (además de por ser árboles binarios de búsqueda) porque sus nodos deben ser de un color (rojos o negros) y además deben cumplir una serie de condiciones de equilibrio:

- La raíz debe ser negra.
- Todo nodo rojo tiene un padre negro.
- Todos los caminos desde la raíz hasta las hojas tienen el mismo número de nodos negros.

Se diseñó una animación de seguimiento completo en la que el alumno puede encontrar todo lo necesario para entender bien el funcionamiento de este tipo particular de árboles, desde su concepto hasta los cambios que hacen falta para que el árbol mantenga sus condiciones de equilibrio (cambios de color y rotaciones).

La animación presenta inicialmente la definición de árbol rojinegro y a continuación se muestra como resolver los casos en que al hacer alguna inserción el árbol pierda sus condiciones de equilibrio. Posteriormente se muestra un ejemplo de inserción de una sucesión de nodos. Todo el ejemplo está totalmente comentado y el alumno dispone en todo momento de un botón que le traslada al concepto teórico de la instrucción que se está ejecutando en cada paso. La intención de esto es conseguir que el alumno pueda estudiar y poner en práctica la teoría al mismo tiempo.



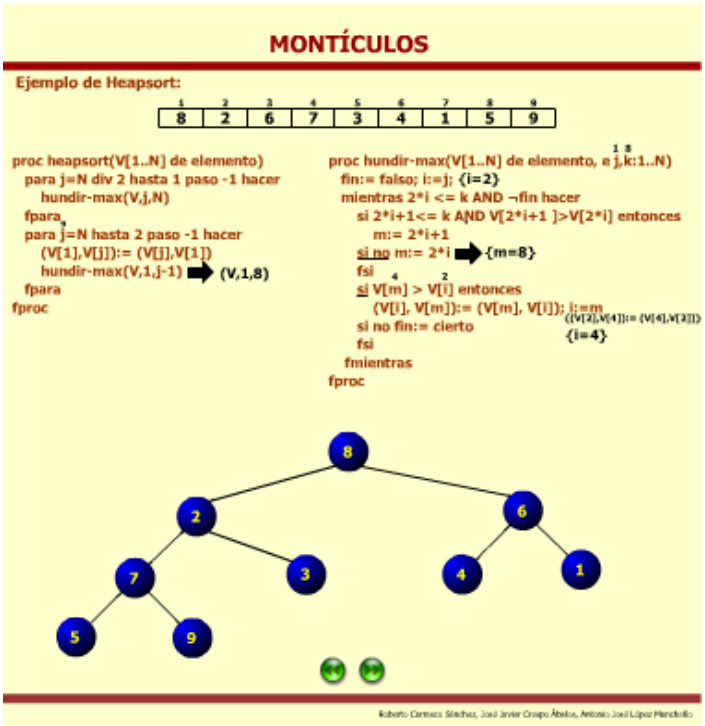
2.4.- Montículos

Un montículo es un caso particular de árboles binarios. Presentan dos características fundamentales, la primera es que son semicompletos* y la segunda es que en función de si son de máximos o de mínimos el elemento de la raíz debe ser mayor o menor que todos los elemento de sus dos subárboles (izquierdo y derecho) y en ambos casos los subárboles son a su vez montículos (de máximos o de mínimos según corresponda).

Por otra parte el método de ordenación heapsort más eficiente que quicksort y mergesort, requiere el conocimiento de montículos para su implementación, por lo tanto, en esta sección la animación diseñada se encarga de enseñar ambos conceptos a los alumnos.

2.4.1 El TAD de los montículos y heapsort

Animación de seguimiento completo. Es una animación con mucha carga teórica (toda ella necesaria para poder entender los conceptos), en ella, en primer lugar se le explica al alumno qué es un montículo y se le ponen ejemplos. Después se le explica lo ventajoso que es ordenar un array por medio del algoritmo heapsort, y se aplica a un ejemplo concreto, desarrollándose paso a paso hasta que el array queda totalmente ordenado. En cada paso el alumno puede comprobar cómo va evolucionando la ordenación del array.



* Un árbol binario se llama semicompleto si las hojas del último nivel ocupan las posiciones más a la izquierda de dicho nivel.

2.5.- Grafos

El TAD o estructura de datos grafos captura el concepto matemático de un grafo y las principales operaciones sobre ellos.

Hoy en día podemos ver muchas cosas que nos pueden parecer de lo más cotidianas, y que sigue una estructura de grafos. Algunos ejemplos son las carreteras, líneas telefónicas, líneas de televisión por cable, el metro, circuitos eléctricos de nuestras casas y otras muchos más.

Definimos un grafo como una de tupla **G = (V,A)**, en donde V y A son conjuntos finitos. Los elementos de V y de A se llaman, respectivamente, **"vértices" y "aristas" de G.**

Esta definición da lugar a una representación gráfica, en donde cada vértice es un punto del plano, y cada arista es una línea que une a sus dos vértices.
Las aristas son las líneas con las que se unen los nodos de un grafo y con la que se construyen también caminos.

El TAD grafos, posee distintas funciones para crear los grafos, estas funciones son:

- Crear: función que crea un grafo vacío.
- InsertarArco: función que dado un grafo y dos vértices de ese grafo, crea una arista entre esos 2 vértices.
- InsertarNodo: función que dado un grafo y un nodo, introduce ese nodo en el grafo.
- BorrarNodo: función que dado un grafo y un nodo, borra ese nodo del grafo
- BorrarArista: función que dado un grafo y dos nodo, borra la arista que une esos dos nodos.

Ejemplo:

G1= InsertarArco(InsertarArco(InsertarNodo(InsertarNodo(Crear,T1),T2),T1,T2),T2,T1)
G2= BorrarNodo(InsertarArco(InsertarNodo(G1,T3), T3,T1),T1)
G3= InsertarArco(InsertarArco(InsertarArco(InsertarNodo(G2,T4),T2,T4),T4,T3),T2,T4)

Los grafos se pueden clasificar en dos grupos: dirigidos y no dirigidos. En un grafo no dirigido el par de vértices que representa una arista no está ordenado. Por lo tanto, los pares (v1, v2) y (v2, v1) representan la misma arista. En un grafo dirigido cada arista está representada por un par ordenado de vértices, de forma que (v1,v2) y (v2,v1) representan dos arcos diferentes.

Los vértices son los puntos o nodos con los que esta conformado un grafo. Llamaremos grado de un vértice al número de aristas de las que es extremo.

Se dice que 2 vértices son adyacentes si tenemos un par de vértices de un grafo (U, V) y si tenemos un arista que los une.

Sean dos vértices X,Y, se dice que hay un camino en G de X a Y si existe una sucesión finita no vacía de aristas {x,v1}, {v1,v2},..., {vn,y}. En este caso
x e y se llaman los extremos del camino.

Entre otras características, de los grafos tenemos:

1. El número de aristas del camino se llama la longitud del camino.
2. Si los vértices no se repiten el camino se dice propio o simple.
3. Si hay un camino no simple entre 2 vértices, también habrá un camino simple entre ellos.
4. Cuando los dos extremos de un camino son iguales, el camino se llama circuito o camino cerrado.
5. Llamaremos ciclo a un circuito simple
6. Un vértice a se dice accesible desde el vértice b si existe un camino entre ellos. Todo vértice es accesible respecto a si mismo

Algunos de los principales tipos de grafos son los que se muestran a continuación:

Grafo regular: Aquel con el mismo grado en todos los vértices. Si ese grado es k lo llamaremos k-regular.

Grafo completo: Aquel con una arista entre cada par de vértices. Un grafo completo con n vértices se denota Kn.

Grafos Isomorfos: Dos grafos son isomorfos cuando existe una correspondencia biunívoca (uno a uno), entre sus vértices de tal forma que dos de estos quedan unidos por una arista en común.

Grafos Platónicos: Son los Grafos formados por los vértices y aristas de los cinco sólidos regulares (Sólidos Platónicos), a saber, el tetraedro, el cubo, el octaedro, el dodecaedro y el icosaedro.

Grafos Conexos:
Un grafo se puede definir como conexo si cualquier vértice V pertenece al conjunto de vértices y es alcanzable por algún otro. Otra definición que dejaría esto más claro sería: "un grafo conexo es un grafo no dirigido de modo que para cualquier par de nodos existe al menos un camino que los une".

Árboles:
Un árbol se define como un tipo de grafo que no contiene ciclos, es decir es un grafo acíclico, pero a su vez es conexo.

Bosques de árboles:
Los bosques de árboles son un caso similar a los árboles, son acíclicos, pero no son conexos.

Recorrido de un grafo.
Recorrer un grafo significa tratar de alcanzar todos los nodos que estén relacionados con uno que llamaremos nodo de salida. Existen básicamente dos técnicas para recorrer un grafo: el recorrido en anchura; y el recorrido en profundidad.

- **Recorrido en anchura**: El recorrido en anchura supone recorrer el grafo, a partir de un nodo dado, en niveles, es decir, primero los que están a una distancia de un arco del nodo de salida, después los que están a dos arcos de distancia, y así sucesivamente hasta alcanzar todos los nodos a los que se pudiese llegar desde el nodo salida.

  Recorrido en profundidad: el recorrido en profundidad trata de buscar los caminos que parten desde el nodo de salida hasta que ya no es posible avanzar más. Cuando ya no puede avanzarse más sobre el camino elegido, se vuelve atrás en busca de caminos alternativos, que no se estudiaron previamente. Dependiendo de cuando se visita el nodo raíz: al visitarlo, después de visitar el hijo izquierdo o después de visitar sus dos hijos, obtenemos tres recorridos diferentes: en preorden, inorden y postorden.

Representación de grafos en programas.

En la práctica se utilizan tres maneras de representar un grafo: mediante matrices, mediante listas y mediante matrices dispersas.

- **Representación mediante matrices**: La forma más fácil de guardar la información de los nodos es mediante la utilización de un vector que indexe los nodos, de manera que los arcos entre los nodos se pueden ver como relaciones entre los índices. Esta relación entre índices se puede guardar en una matriz, que llamaremos de adyacencia. Si el grafo es no valorado, la matriz es de ceros y unos. Si el grafo es valorado, la matriz almacena el coste del camino.

- **Representación mediante listas**: En las listas de adyacencia se guarda por cada nodo, además de la información que pueda contener el propio nodo, una lista dinámica con los nodos adyacentes. La información de los nodos se puede guardar en un vector, al igual que antes, o en otra lista dinámica,

- **Representación mediante matrices dispersas**: Para evitar uno de los problemas que existentes con las listas de adyacencia, que es la dificultad de obtener las relaciones inversas, podemos utilizar las matrices dispersas, que contienen tanta información como las matrices de adyacencia, pero, en principio, no ocupan tanta memoria como las matrices, ya que al igual que en las listas de adyacencia, sólo representaremos aquellos enlaces que existen en el grafo.

Algoritmos importantes aplicados a grafos

A continuación, mostramos los algoritmos más usuales aplicados al TAD de grafos para conseguir el árbol de recubrimiento mínimo, que se define como un subgrafo que:

- Contiene todos los vértices del grafo
- Es conexo
- El coste total de las aristas es mínimo
- Para calcular el camino mínimo entre dos nodos del grafo

Los algoritmos a analizar son los siguientes:

- Algoritmo de Dijkstra: para el cálculo de caminos mínimo
- Algoritmo de Kruskal: para el cálculo de árbol de recubrimiento mínimo
- Algoritmo de Prim: para el cálculo del árbol de recubrimiento mínimo

2.5.1 Árbol de recubrimiento. Algoritmo de Prim

El algoritmo de Prim es un algoritmo de la teoría de los grafos para encontrar un árbol de recubrimiento mínimo en un grafo conexo y valorado. En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices del grafo, en donde la suma del peso total de todas las aristas en el árbol es el mínimo posible. El algoritmo fue diseñado en 1930 por el matemático Vojtech Jarnik y luego de manera independiente por el científico computacional Robert C. Prim en 1957 y redescubierto por Dijkstra en 1959. Por esta razón el algoritmo es también conocido como algoritmo DJP o algoritmo de Jarnik.

Implementación

```
fun Prim (G:grafo) dev T: Conjunto [aristas]
{
        V:= escoger_vertice(G);
        T:= {};
        W:= vértices(G) – {v};
        U:= {v};
        mientras no_vacio?(W) hacer
        {
                (u,v) := ArcoCosteMinimo(u,v);
                T:= T u {(u,v)};
                U:= U u {v};
                W:=W – {v};
        }
return T
}
```

Procedimiento del algoritmo:

El algoritmo se puede implementar con matrices de adyacencia y con listas de adyacencia. La animación muestra el algoritmo en pseudocódigo, independiente de la representación elegida:

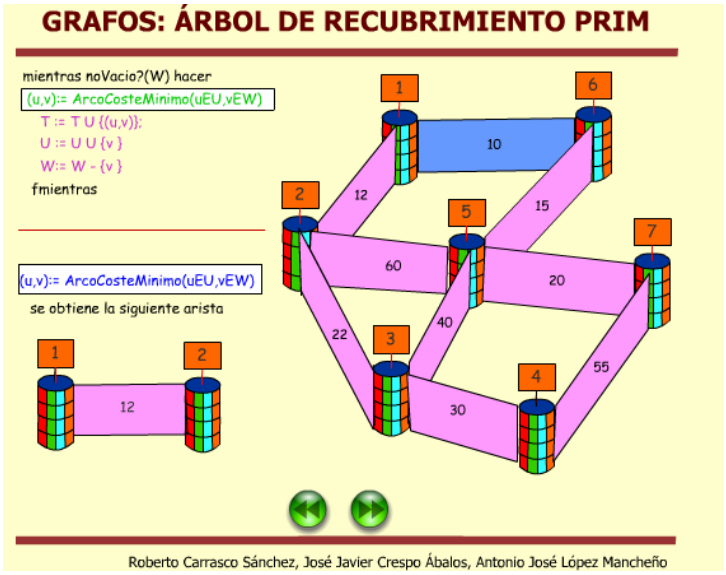
Disponemos de 2 funciones:

Escoger_vertice(G), que escoge un vértice entre todos los vértices del grafo.
arcoCosteMinimo(U,W), que devuelve la arista de menor coste de entre todas las aristas formadas por un vértice del conjunto U y otro vértice del conjunto W.

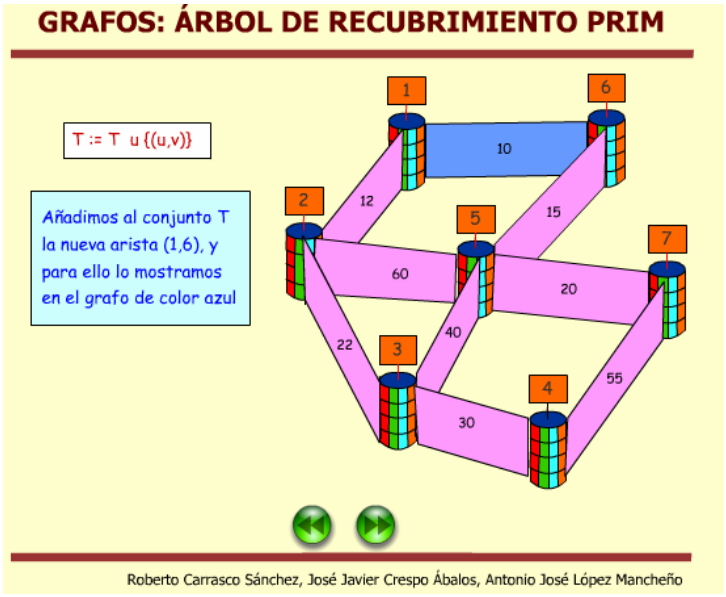
Utilizaremos las variables:
T, que es un conjunto que representa árbol de recubrimiento resultante.
U, conjunto en el que se almacenan los nodos ya visitados.
W, conjunto en el que se almacenan los nodos que quedan por visitar.

La raíz del árbol es un vértice cualquiera del grafo.

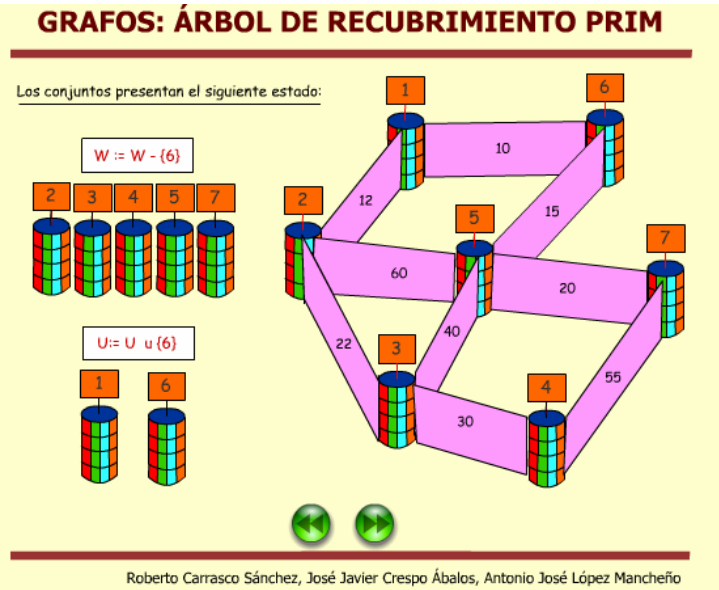
Al comienzo se selecciona la raíz del árbol.
A continuación mientras queden vértices por incorporar:
Enumerar la arista con menor coste entre los conjuntos U y W.



Se incorpora al conjunto T la arista que menos le cueste.

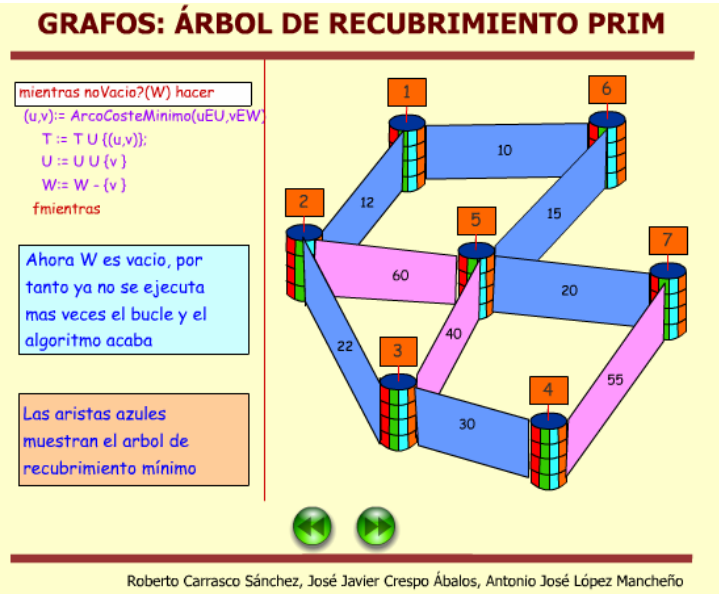


Se actualizan los conjuntos U y W.



Se repite el proceso hasta que el conjunto W sea vacío.

El resultado final será:



Complejidad del algoritmo

Para calcular la eficiencia, supongamos que sea n el número de vértices del grafo y m el número de aristas.

El coste de obtener el vector de aristas es $O(n^2)$ cuando se utilizan matrices de adyacencia para representar el grafo (el coste sería el mismo si

hubiéramos utilizado listas de adyacencias, ya que en este algoritmo el coste es independiente de la representación).

2.5.2 Árbol de recubrimiento. Algoritmo de Kruskal

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol de recubrimiento mínimo en un grafo conexo y valorado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el peso total de todas las aristas del árbol es el mínimo. El algoritmo de Kruskal es un ejemplo de algoritmo voraz.

Implementación

```
Fun Kruskal (G:grafo) dev T: Conjunto[aristas]
{
        Partición P;
        T:= {};
        Mientras (numAristas(T) <= n-1) hacer
        {
                (u,v) := ArcoCosteMinimo(G);
                Conjunto C := P.buscar(u);
                Conjunto D := P.buscar(v);
                If(C!=D)
                {
                        P.fusionar(C,D);
                        T := T u {(u,v)};
                }
        }

Return T;
}
```

Procedimiento del algoritmo:

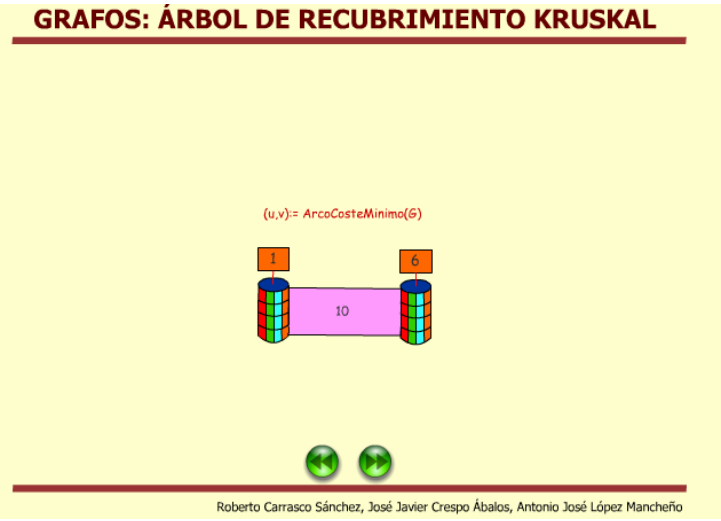
Suponemos los vértices fijados y representados por naturales 0..N-1.

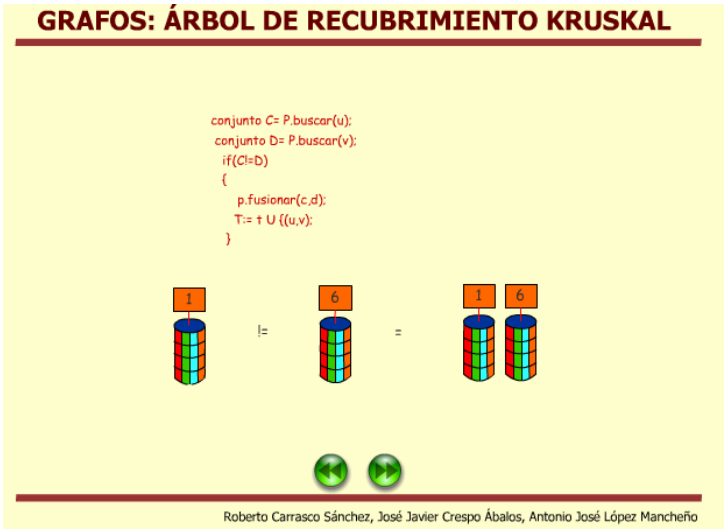
Parte de un bosque de árboles formados por un único vértice y para ello, creamos las particiones unitarias.

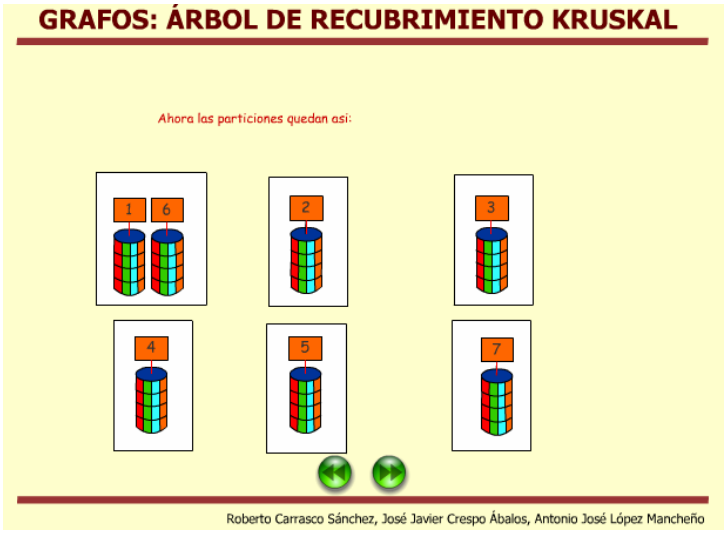
Se utiliza un tipo de datos partición sobre los vértices con la relación de equivalencia, ser_adyacente. Las operaciones que proporciona el tipo de datos son:

buscar la clase de equivalencia a la que pertenece un elemento.
fusionar en una partición las clases de equivalencia correspondientes a dos elementos a y b del conjunto, al añadir el par (a,b) a la relación.

Mientras que en el bosque haya más de un árbol se selecciona la arista de menor coste que no forme ciclos en el conjunto de aristas seleccionado previamente.

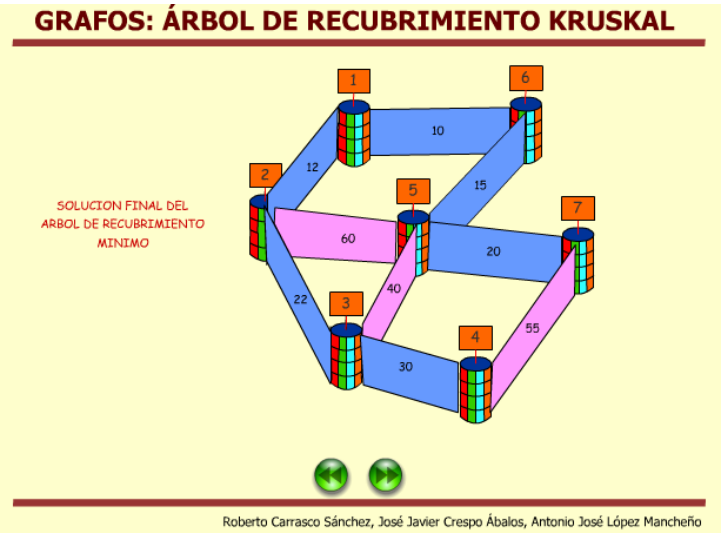






Se dispone de las siguientes funciones:
numArista(T), que devuelve el número de aristas que tiene el conjunto T.
arcoCosteMinimo(G) que devuelve la arista de menor coste del grafo G.

El algoritmo devuelve T, que es un conjunto de aristas que representa el árbol de recubrimiento mínimo.



Complejidad del algoritmo

Para calcular la eficiencia, supongamos que sea n el número de vértices del grafo y m el número de aristas.

El coste de obtener el vector de aristas es O(n2) cuando se utilizan matrices de adyacencia para representar el grafo y de O(n + m) cuando se utilizan listas de adyacencia.

El coste de ordenar un vector de aristas está en O(m log(m)).

El coste de crear la partición es O(n).

El bucle realiza en el caso peor m iteraciones, con 2m búsquedas en la partición y n-1 fusiones. Con una buena implementación del TADF partición el coste de las 2m + (n-1) operaciones es prácticamente lineal.

El coste del algoritmo proviene de la fase de inicialización y está en el orden de $O(n^2)$ en el caso de una representación mediante matrices de adyacencia y $O(n^2)$ en el caso de una representación mediante matrices de adyacencia y O(m log(m)) en el caso de una representación con listas de adyacencia

2.5.3 Cálculo de caminos mínimos. Algoritmo de Dijkstra

También llamado algoritmo de caminos mínimos, es un algoritmo para calcular el camino mínimo desde un vértice a todos los demás en un grafo conexo, dirigido y valorado.. Su nombre se refiere a Edsger Dijkstra, quien lo describió en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo se atiene a la estrategia conocida como esquema de algoritmo voraz.

Implementación con matrices de adyacencia

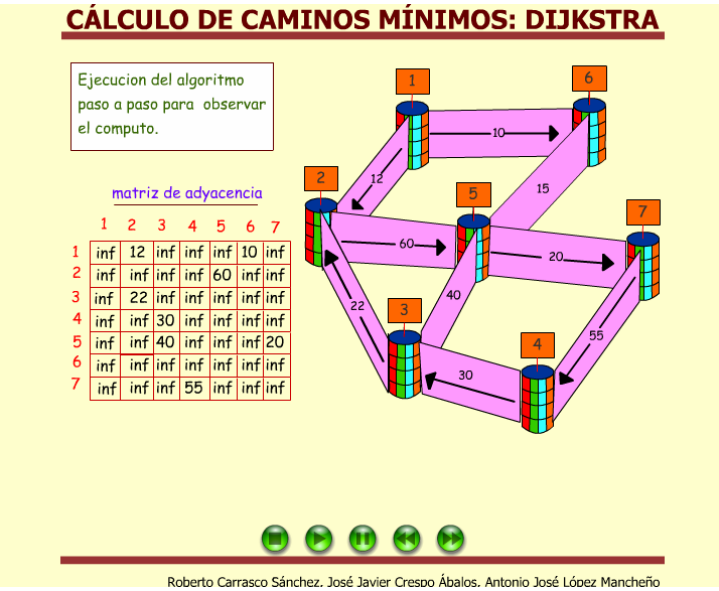
```
Fun Dijkstra (G: grafo) dev
var
        Candidatos: conjunto[2..n];
        Coste_min [1..n] de valor;
        Predecesor[2..n] de 1..n;

        para i=2 hasta n hacer
        {
        añadir(i, candidatos);
        coste_min[i] = G[1,i];
        predecessor[i] = 1;
        }

        para i=1 hasta n-2 hacer
        {
        minimo = maxint;
        para j=2 hasta n hacer
                {
                Si( esta?(j, candidatos) && (coste_min[j] < minimo)  )
                        {
                        minimo = coste_min[j];
                                elegido = j;
                }
                        quitar(elegido, candidatos)
                para j=1 hasta n hacer
                        {
                                Si( coste_min[elegido] + G[elegido, minimo]<
                                                                Coste-min[j]
                                                                    )
```

```
                    {
                    coste_min[j] = coste_min[elegido] +
                                        G[elegido, j];
                        Predecesor[j] = elegido;
                }
            }
        }
```

Procedimiento del algoritmo:

Para representar el grafo, utilizaremos la matriz de adyacencia, donde:

- Matriz[i,j] = c si existe arista entre los nodos i y j y el coste es c
- Matriz[i,j] = inf si no existe arista entre los nodos i y j



El conjunto de vértices se divide en dos subconjuntos: los seleccionados, para los que ya se conoce el coste mínimo y los no seleccionados, para los que se conoce el coste del camino mínimo atravesando únicamente vértices seleccionados.

Inicialmente el único vértice seleccionado es el origen.

En cada etapa se selecciona el vértice que tenga menor coste desde el origen y se actualiza el coste desde el origen a los vértices restantes.

Los datos de salida son:

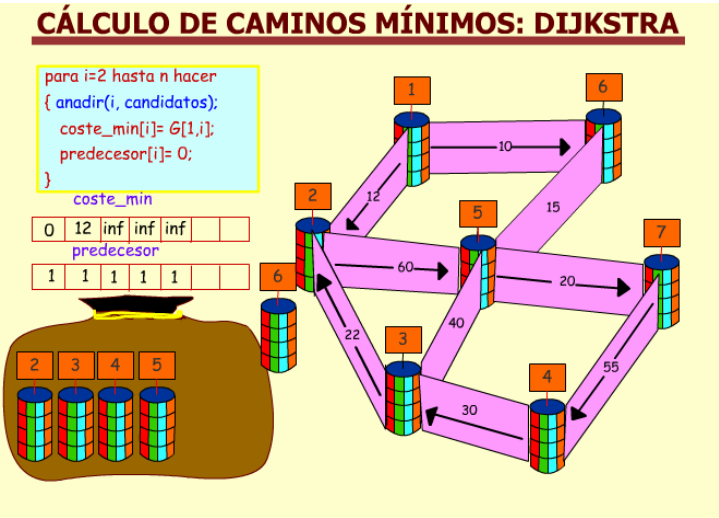
- El vector coste_min[] con el coste del camino seleccionado.
- El vector predecesor[] con el predecesor de cada nodo en el camino desde el origen hasta el nodo.

Suponemos que los vértices están enumerados del 1..N. Por lo que el grafo está representado únicamente por su matriz de adyacencia.

El origen es el primer vértice 1.

Para obtener en cada paso del algoritmo el vértice cuyo camino desde la raíz tiene coste mínimo recorremos el vector coste_min. El conjunto candidatos nos indica en cada momento los vértices del vector coste_min que todavía no han sido seleccionados.

En la parte inicial del algoritmo añadimos a candidato, representado en la animación como un saco, todos los nodos del grafo excepto el primero y actualizamos coste_min y predecesor a partir de la matriz de adyacencia.

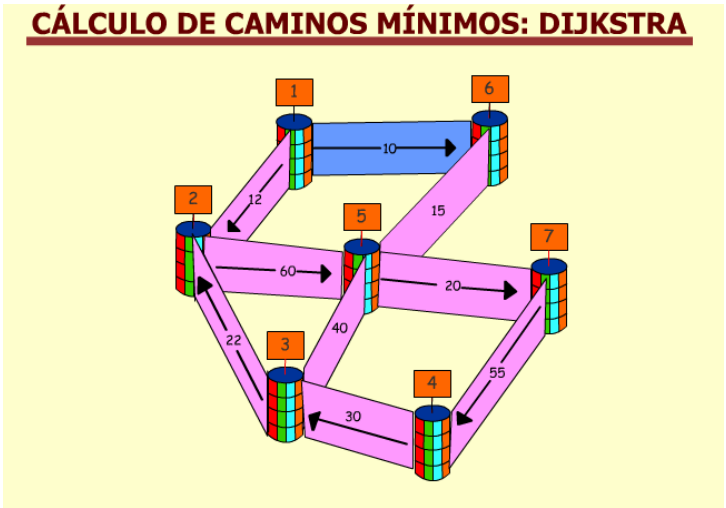


La parte principal del algoritmo, que es el para que recorre todos los nodos, se divide a su vez, en 3 bloques.

El primer bloque es para obtener el nodo con coste minimo(elegido y coste_min) de todos los nodos presentes en el saco.

La segunda parte quita de candidatos, o del saco en la animación, el nodo elegido.

Se obtiene así la arista buscada.



La tercera parte actualiza los vectores coste_min y predecesor según las condiciones actuales.

Complejidad

El tamaño del problema n, respecto al que calculamos la complejidad será el número de vértices del grafo

Puesto que en cada iteración eliminamos un vértice del conjunto candidatos, que está inicializado con el número total de vértices, está claro que se ejecuta un total de n iteraciones.

A su vez, otro bucle que se ejecuta una vez para cada uno del resto de vértices que quedan en candidatos en ese instante, para los cuales se recalcula el camino mínimo conocido. Esta operación toma un tiempo constante, pero como se ejecuta un número de veces n-i, tenemos que le tiempo total es

$$T(n) = \sum_{i=1}^{n} n - i = \sum_{i=1}^{n} i = O(n^2)$$

3.- Implementación de las animaciones

Las animaciones se han creado en formato flash (Shockwave Flash -.swf-) mediante la herramienta "Macromedia Flash MX 7". Utilizamos esta herramienta de desarrollo por dos motivos, en primer lugar porque es una herramienta potente para desarrollar animaciones del estilo de las que eran el objetivo de este proyecto y en segundo lugar porque los archivos swf pueden reproducirse tanto desde un navegador (mediante un plugin que ya va incorporado en casi todos los navegadores actuales y si no fuera el caso se puede conseguir mediante una simple actualización por Internet) como desde la aplicación que reproduce los archivos swf (Macromedia Flash Player) que se puede obtener gratuitamente en `www.macromedia.com` aunque ya hoy en día está incluido en los sistemas operativos Windows XP, Apple Macintosh y Linux. Según Macromedia los archivos swf se pueden reproducir desde un 98% de los equipos de escritorio con conexión a Internet de todo el mundo y en una amplia gama de dispositivos populares.

Además, los archivos swf no son editables (los archivos editables tienen extensión .fla, que luego darán lugar a los .swf), esto unido a que las películas flash comienzan a reproducirse inmediatamente y no es necesario descargar el archivo completo para poder empezar a verlo (técnica denominada streaming, que hace accesibles las animaciones desarrolladas en flash incluso a través de conexiones lentas) y que puede ser reproducido por casi cualquier navegador quiere decir que tenemos unas animaciones que se visualizan correctamente con independencia del navegador o el computador utilizado y por lo tanto estamos trabajando en un formato muy potente para ser mostradas desde una Web dedicada a la enseñanza de estructuras de datos (podrán ser accesibles prácticamente para todo el mundo que disponga de conexión a Internet).

Otro aspecto reseñable de los archivos swf es que pueden incluir todo tipo de elementos gráficos, textos, sonido y vídeo y ajustan automáticamente el tamaño del texto y los gráficos para adaptarse al tamaño de la ventana del navegador. Además del aspecto meramente estético, Flash introduce en su entorno la posibilidad de interaccionar con el usuario. Para ello, Flash invoca un lenguaje de programación llamado ActionScript. Orientado a objetos, este lenguaje tiene claras influencias del Javascript y permite, entre otras muchas cosas, gestionar el relleno de formularios, ejecutar distintas partes de una animación en función de eventos producidos por el usuario, saltar a otras páginas, etc.

De este modo, Macromedia pone a nuestra disposición una tecnología pensada para aportar vistosidad a nuestra Web al mismo tiempo que nos permite interaccionar con nuestro visitante. Por supuesto, no se trata de la única alternativa de diseño aplicada al Web pero, sin duda, se trata de la más popular y más completa de ellas.

3.1 Aspectos básicos de una animación en Macromedia Flash

Aprender a hacer animaciones con este programa requiere por una parte un conocimiento del software y por otra, una aplicación inteligente de los recursos que se nos ofrece. Vamos a explicar los aspectos básicos que están presentes en cualquier animación generada mediante Macromedia Flash, de este modo conseguiremos un doble objetivo: dar una información inicial a las personas que quieran continuar este proyecto o a las personas interesadas en manejar esta aplicación, y por otra parte mostramos en qué ha consistido una parte del trabajo desarrollado.

En cierta medida, Flash trabaja como si fuese una película. Una animación es una sucesión de imágenes fijas que, al pasar rápidamente unas detrás de otras, dan la impresión de que existe movimiento. Cada una de estas imágenes fijas es llamada también fotograma.

Por otra parte, una animación ésta generalmente constituida por una variedad de objetos diferentes, cada uno de los cuales se introduce en un momento diferente y presenta un comportamiento independiente al resto de los objetos. Con el motivo de organizar y editar todos estos elementos Flash permite el uso de capas.

Así, una animación Flash esta compuesta de una superposición de capas en cada una de las cuales introduciremos un objeto que tendrá su propia línea de fotogramas. Estas capas nos permiten trabajar la animación en distintos planos independientes. Las capas actúan como una serie de hojas transparentes superpuestas que permiten mantener todos los componentes de los fotogramas por separado para poder manipularlos de forma individual.

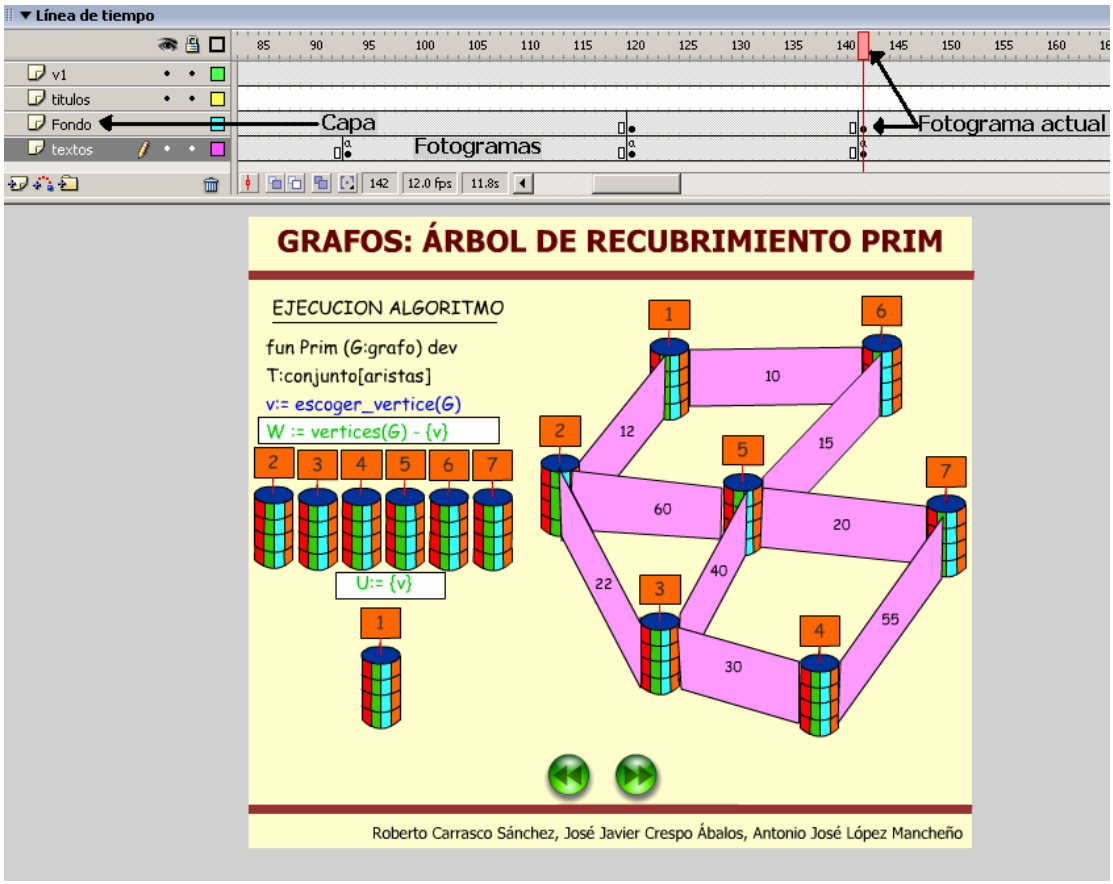
Por defecto, al comenzar una nueva escena encontraremos en nuestra línea de tiempo una sola capa. Progresivamente iremos introduciendo más capas que nos permitan separar cada uno de los elementos de la animación: objetos, fondo, sonidos o trayectorias.

Un uso inteligente de las capas es la base para crear animaciones de calidad.

Como ya hemos dicho, en Flash, la animación se crea cambiando el contenido de fotogramas sucesivos. Lo cual puede llegar a generar movimientos de todo tipo. Se puede hacer que un objeto se desplace a lo largo de un recorrido, aumente o disminuya de tamaño, gire, cambie de color, aparezca o desaparezca, y cambie de forma. Estas transformaciones pueden ocurrir por separado o combinadas entre sí. Por ejemplo, se puede hacer que un objeto gire al mismo tiempo que aparece y se desplaza.

Flash ofrece dos maneras de crear secuencias de animación: fotograma a fotograma y por interpolación. En la animación fotograma a fotograma se

crea una imagen distinta en cada uno de ellos. En la animación por interpolación se crean los fotogramas inicial y final, y Flash se encarga de crear los fotogramas intermedios. Este último tipo, obviamente no siempre puede ser utilizado, ya que exige que el movimiento siga una trayectoria recta, que la figura no experimente cambio alguno y además sólo puede ser desplazado todo el contenido de una capa a la vez.



Flash es la tecnología más comúnmente utilizada en el Web que permite la creación de animaciones vectoriales. El interés en el uso de gráficos vectoriales* es que éstos permiten llevar a cabo animaciones de poco peso, es decir, que tardan poco tiempo en ser cargadas por el navegador. Y es que Flash es un editor de gráficos vectoriales similar a programas como Illustrator, Corel Draw, o Freehand, pero a diferencia de ellos ha sido diseñado con el objetivo de facilitar la creación de animaciones y para añadir interactividad a las páginas Web.

* Existen dos tipos de gráficos. Por una parte están los *gráficos vectoriales*, en los cuales una imagen es representada a partir de líneas (o vectores) que poseen determinadas propiedades (color, grosor…). La calidad de este tipo de gráficos no depende del zoom o del tipo de resolución con el cual se esté mirando el gráfico. Y por otra parte están las *imágenes en mapa de bits*, que se asemejan a una especie de cuadrícula en la cual cada uno de los cuadrados (píxeles) muestra un color determinado. Este tipo de gráficos son dependientes de la variación del tamaño y resolución, pudiendo perder calidad al modificar sucesivamente sus dimensiones.

Cuando finalicemos la película Flash y queramos mostrarla al público, debemos publicar o exportar el archivo Flash .fla a otro formato para su reproducción. Macromedia Flash tiene una opción diseñada para presentar la animación en una página Web, y se encarga de crear automáticamente el archivo de Flash Player (swf) y un documento HTML que inserta dicho archivo para poder ser visualizado por cualquier navegador.

3.2 Biblioteca de símbolos

En la entrega en formato electrónico que se hace de este proyecto se adjuntará en forma de fichero .fla una *biblioteca de símbolos*, con el objeto de facilitar la continuidad de este proyecto.

Para que el lector de este texto pueda entender la "herencia" que le dejamos a futuros compañeros debemos explicar qué es un símbolo y qué es una biblioteca de símbolos.

Una biblioteca no es más que un almacén de objetos (gráficos o sonidos) que podrán ser utilizados en distintas animaciones (o en la misma animación) tantas veces como se desee.

Cada uno de los elementos que constituyen una biblioteca se denominan símbolos. Como hemos dicho, estos elementos podrán ser utilizados en nuestra animación cuantas veces lo deseemos.

3.3 El lenguaje ActionScript de Macromedia Flash

ActionScript es el lenguaje de creación de scripts o guiones de Flash, y se puede utilizar para controlar objetos en las películas de Flash, crear elementos interactivos y de navegación, elaborar juegos, formularios, encuestas y también para desarrollar todo tipo de aplicaciones Web.

Las películas se pueden configurar para que ejecuten scripts que respondan a eventos de usuario, como puede ser la pulsación del botón del ratón o de una tecla determinada. En ciertos aspectos ActionScript es un lenguaje de programación similar a JavaScript[*].

[*] JavaScript es un lenguaje de programación con el que se pueden crear páginas Web interactivas en las que existe una comunicación entre los componentes de la página, el estado del navegador y las operaciones realizadas por los usuarios. Esto significa que con JavaScript podemos conseguir que nuestras Web interactúen con los usuarios y respondan a sus acciones. Desde su aparición se ha convertido en uno de los mejores complementos del lenguaje HTML porque permite que las páginas Web realicen algunas tareas por sí mismas, sin necesidad de comunicarse con el servidor en el que están almacenadas.

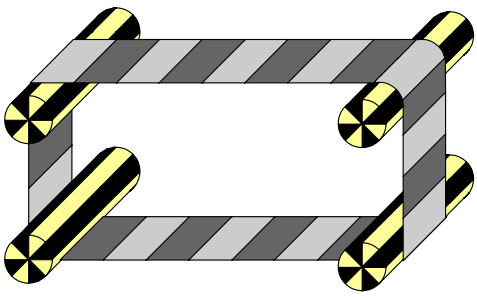
ActionScript es un lenguaje de script, esto es, no requiere la creación de un programa completo para que la aplicación alcance los objetivos marcados.

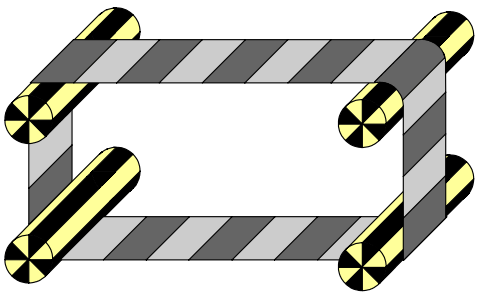
Todas las animaciones de este proyecto presentan fragmentos de código del lenguaje ActionScript. Estos fragmentos de código son los que permiten programar las acciones que se ejecutarán al pulsar sobre los botones de las animaciones.

3.4.- Problemas surgidos y soluciones tomadas

El primer problema que surgió fue el de encontrar un software que nos permitiera desarrollar animaciones del estilo de las que queríamos diseñar. Encontramos programas capaces de desarrollar animaciones flash (Shockwave Flash) gratuitos y más sencillos de manejar de Macromedia Flash, pero tenían un fallo esencial, y es que no eran nada potentes, es decir, sólo facilitaban movimientos simples y no presentaban elementos tan necesarios como bibliotecas de componentes, además, habría sido una pérdida de tiempo haber aprendido a manejar un software nada conocido en lugar de Macromedia Flash, que es el más usado para este tipo de trabajos.

El siguiente problema fue el de crear movimiento a la cinta corredera (ejemplo 2.1.3), teníamos que ser capaces de conseguir que una cinta del estilo a las que transportan las maletas en los aeropuertos cobrara movimiento. Para ello buscamos en Internet información relacionada con los dibujos animados, y cómo se conseguía crear efectos de movimiento con los mismos a partir de imágenes estáticas. A partir de varias ideas terminamos decidiendo que lo que había que hacer era dividir la cinta en franjas de dos colores diferentes, y cada cierto tiempo (unos 5 fotogramas) cambiar una cinta por otra que tenía los colores de sus franjas invertidos. Así, se consigue que al ojo humano le de la sensación de movimiento, ya que el que visualiza la animación si sigue una franja en particular le dará la impresión de que se desplaza continuamente hasta terminar una vuelta completa sobre la cinta. A continuación mostramos los dos tipos de cintas que tuvimos que diseñar, para ver el resultado final se recomienda ver la animación:





Al igual que en el caso anterior, en algunas ocasiones del desarrollo de las animaciones no hay que preguntarse cómo podemos conseguir movimiento, sino de qué modo podemos engañar al ojo humano para que le de la impresión de que lo hay. Y es que al diseñar en Flash en multitud de ocasiones surgen problemas de diseño o de dibujo que no siempre pueden resolverse. En esos casos hay que ponerse en el lugar del usuario y decidir qué es lo importante, qué es lo que va a pasar inadvertido y actuar en consecuencia. Por ejemplo, las esferas que pueden verse en la animación en la que se define el funcionamiento de de las pilas o de las colas no son figuras tridimensionales, simplemente están coloreadas con una opción de Macromedia Flash que permite iluminar más una zona que otra de una circunferencia, provocando de este modo al ojo humano la sensación de estar viendo una esfera.

Como dijimos en la introducción las animaciones debían cumplir tres objetivos distintos:

- Ser fáciles de comprender por el alumno.
- Presentar las características necesarias para poder ser expuestas en público.
- Dotarlas de una gran carga visual para poder ser proyectadas en aulas de grandes dimensiones.

El primer y el último objetivo se han enfocado a partir de los comentarios de la directora del proyecto, que nos ha ayudado haciéndonos ver qué métodos o técnicas ayudan más a los alumnos a entender lo que les quiere transmitir. Y en cuanto al segundo se puede decir que está cubierto por el sistema de navegación que presentan todas las animaciones.

3.5.- Resumen técnico

Una vez terminadas todas las animaciones del proyecto hemos hecho un resumen técnico sobre el mismo, en el que se han contado animación tras animación todas las capas y los fotogramas que se han necesitado, así, la suma total de capas asciende a 307 y la de fotogramas a 236.231. Obviamente no hemos contabilizado cuantos de esos 236231 son claves y cuántos no los son, habría sido una tarea tan tediosa como inútil.

El desglose total por capas y fotogramas es el siguiente:

Animación	Nº Capas	Fotogramas por capa	Producto
2.1.1 El TAD de las pilas	9	186	1674
2.1.2 Transformación de expresiones infijas a postfijas	32	304	9728
2.1.3 Uso en una cinta corredera	16	106	1696
2.2.1 El TAD de las colas	12	166	1992
2.2.2 Uso en una cinta corredera con cola intermedia	17	168	2856
2.3.1 El TAD de los árboles binarios de búsqueda	17	7	119
2.3.2 Recorrido en anchura de un árbol binario	58	1269	73602
2.3.3 Árboles rojinegros	3	30	90
2.4.1 El TAD de los montículos y heapsort	21	65	1365
2.5.1 Árbol de recubrimiento. Algoritmo de Prim	28	790	22120
2.5.2 Árbol de recubrimiento. Algoritmo de Kruskal	34	54	1836
2.5.3 Cálculo de caminos mínimos. Algoritmo de Dijkstra	50	2269	113450

Nº total de fotogramas: 230528

4.- Entorno Web

Debido a la reorganización que sufrió el proyecto en su comienzo, la parte Web del mismo, que comentamos a continuación, quedo mucho mas reducida en contenido y funcionalidades.

La mayor dedicación del proyecto recayó en las animaciones de los distintos tipos de datos y algoritmos. Como muestra de ello basta comentar el total de frames acumulados entre todas que supera los 230.000, lo que demuestra el tiempo invertido en las mismas.

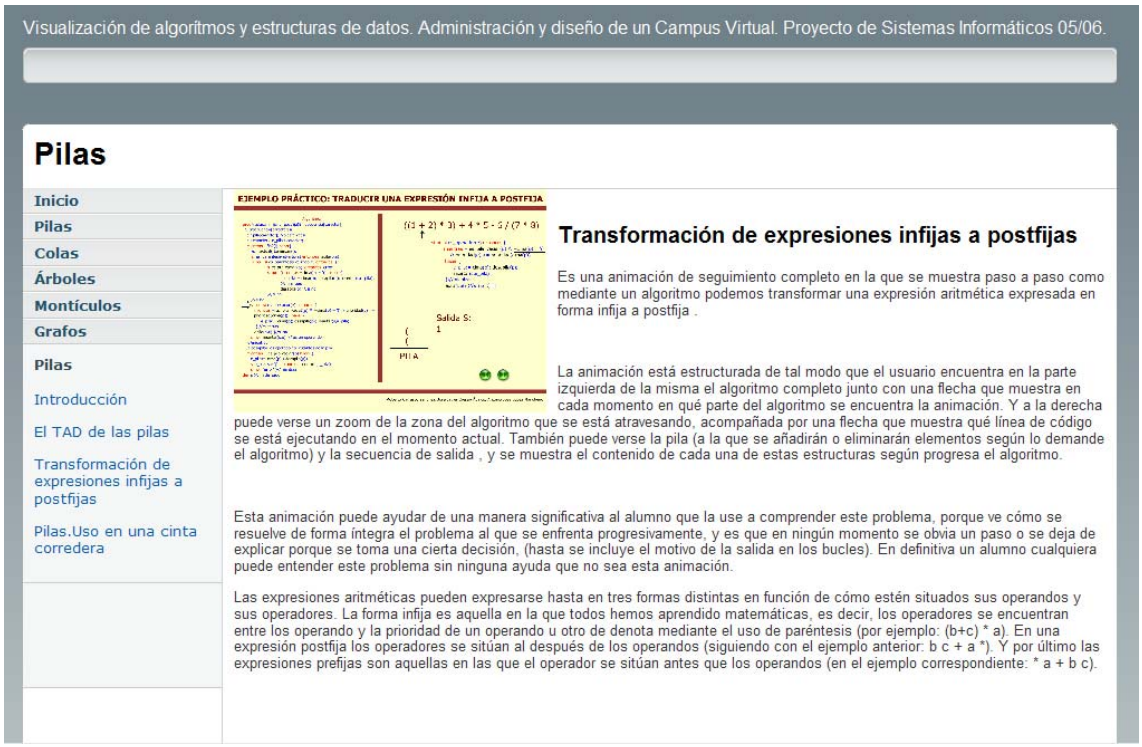
Sin embargo, se ha creado la base del entorno Web, formada por el entorno de paginas Web con los contenidos, la base de datos del proyecto (con las contraseñas para la identificación de los usuarios básicamente) y el servidor de aplicaciones.

En todos los casos se ha optado por herramientas libres. El servidor para la base de datos elegido ha sido MySQL, dado que se adaptaba perfectamente al pequeño tamaño de nuestra base de datos. El servidor de aplicaciones utilizado es el Apache Tomcat.

4.1 Entorno Web

Se ha desarrollado un entorno de páginas Web para mostrar todo nuestro trabajo a través de un navegador, alojando dicho entorno en un servidor de aplicaciones para poder ofrecerlo desde cualquier servidor en el que instalemos el conjunto.

A continuación se presenta una captura de pantalla de una de las páginas.



Para el diseño de las páginas Web se optó por la sencillez y la funcionalidad. No se buscaba un diseño recargado que dificultase la navegación. Se eligió un elegante diseño utilizando hojas de estilo. De esta forma a través de una hoja de estilo creada para nuestro sitio, dotamos a todas las paginas de una apariencia idéntica (algo fundamental) y de un diseño claro que permite localizar todas las secciones con gran rapidez gracias a los intuitivos menús que aparecen en todas y cada una de las paginas, evitando incomodas navegaciones para acceder al menú principal. Además, el diseño basado en nuestra hoja de estilo permite una rápida modificación de elementos básicos del diseño como pueden ser el tamaño del tipo de letra, colores y disposición de los elementos. De esta forma se facilita la modificación del diseño según las necesidades y sobre todo permite incrementar el tamaño del sitio añadiendo nuevas páginas para un futuro contenido con muy poco esfuerzo.

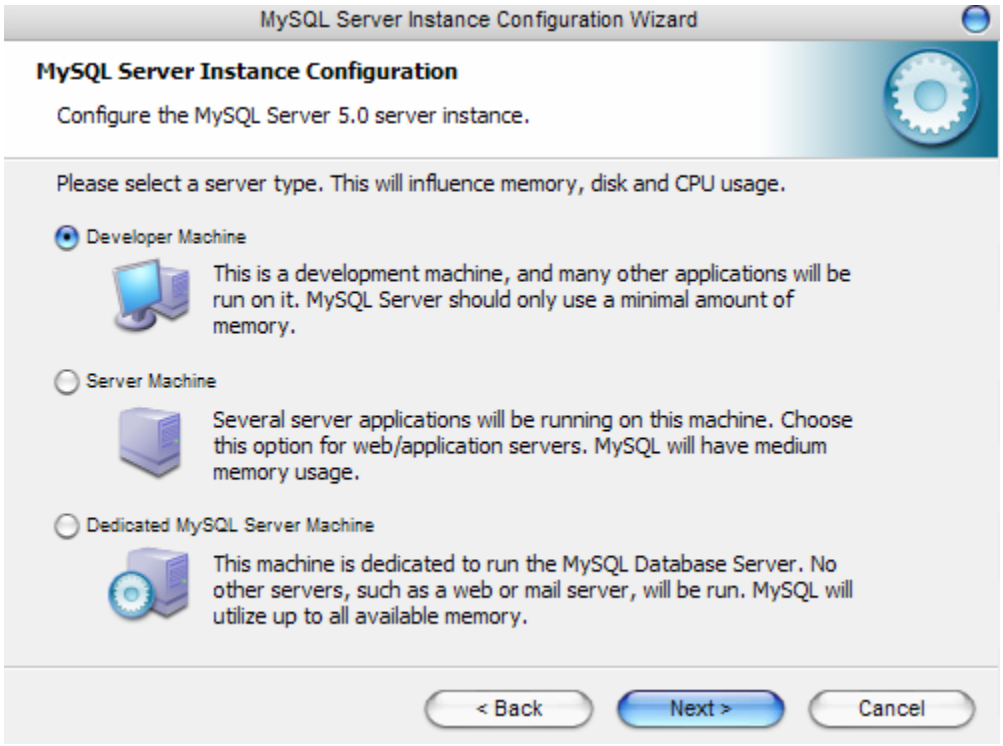
Incluso se pueden añadir directamente nuevas hojas de estilo con parámetros adaptados para personas con problemas visuales y otras necesidades especiales sin necesidad de modificar nada más del entorno.

En la captura de pantalla puede verse el aspecto común de las páginas. Todas presentan un menú izquierdo en columna que permite acceder a cada sección. Dentro de cada sección, existe una página introductoria del tipo de datos en cuestión y los elementos creados. Además cada animación se presenta en su propia página junto con la correspondiente explicación o texto referente a la misma. En todas las páginas que presentan una animación, existe una captura de pantalla de la misma. De esta forma basta con pulsar con el ratón sobre ella para que la animación flash comience en nuestro navegador. Una vez terminado su estudio para volver al entorno de páginas simplemente es necesario pulsar la tecla de “pagina anterior” o “retorno” del navegador que estemos utilizando.

Es necesario remarcar que el diseño del entorno se ha optimizado para el navegador Firefox. En otros navegadores las paginas se podrán ver perfectamente, aunque es posible que algún elemento varíe ligeramente su posición respecto al diseño original y que se conseguiría utilizando Firefox.

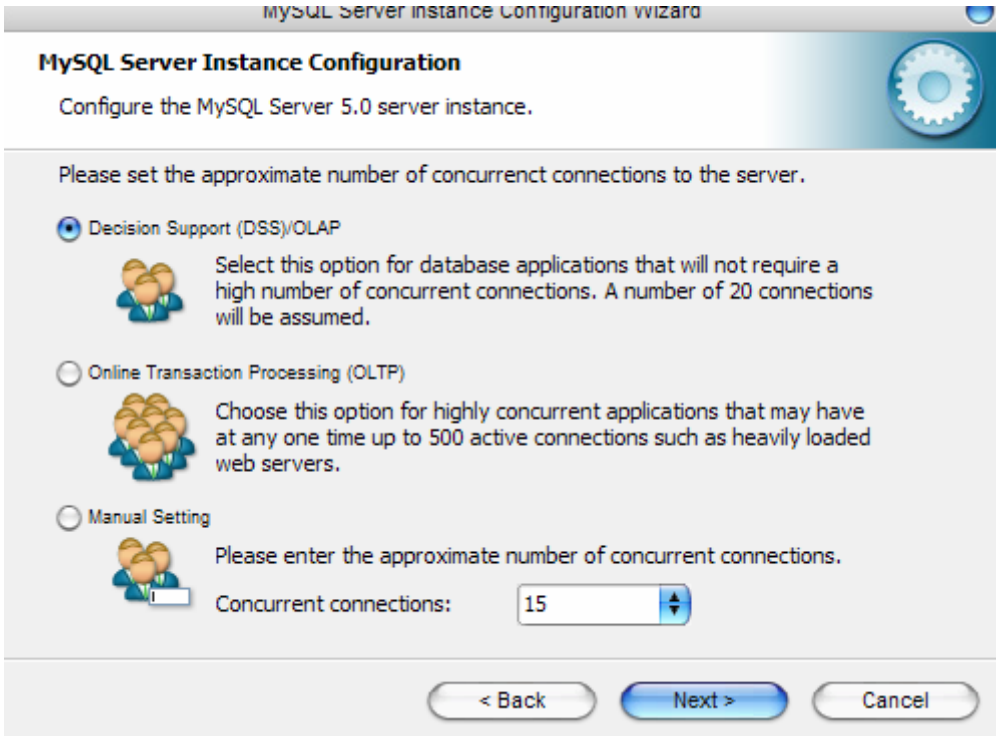
4.2 Base de datos

El servidor de bases de datos utilizado ha sido MySQL, concretamente la versión 5.0.19 para Windows, plataforma en la que se realiza la instalación. En la instalación del servidor es necesario elegir la opción “Custom” que nos permite personalizar ciertos parámetros necesarios para restaurar la sencilla base que se proporciona con el proyecto. No es necesario modificar el directorio de instalación proporcionado por defecto. Es recomendable elegir la siguiente opción de instalación:



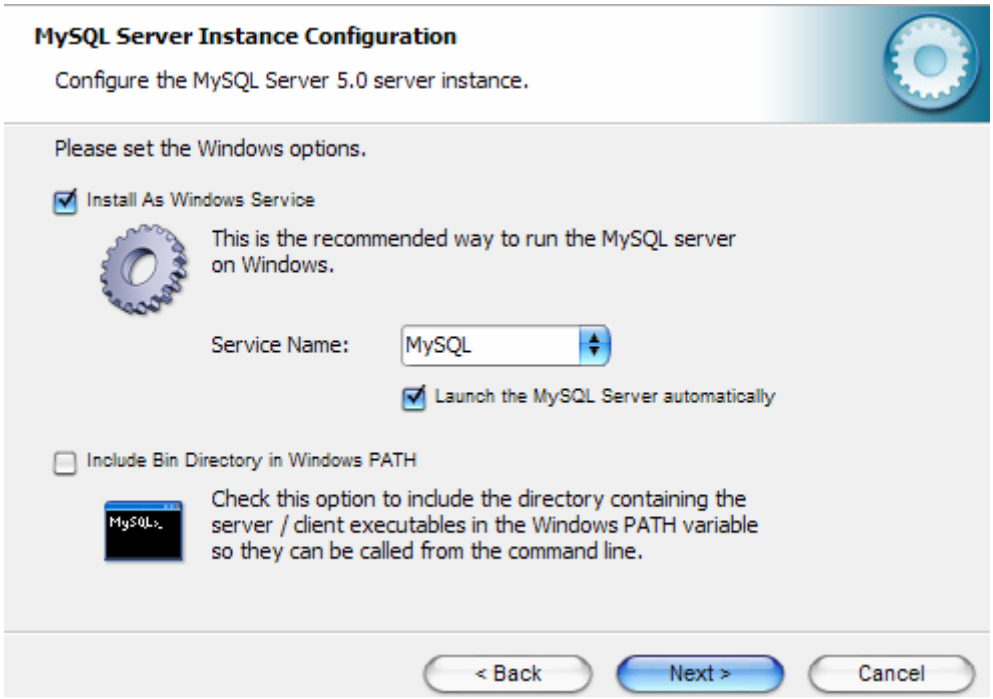
De esta forma estamos indicando que el servidor ocupe la menor memoria necesaria. Es recomendado si estamos utilizando el equipo para otras tareas o estamos ejecutando otras aplicaciones como será nuestro caso con el servidor de aplicaciones Web.

A continuación se solicita elegir el tipo de configuración para la base de datos. Se recomienda seleccionar “Multifuncional Database”. Cuando se nos solicite el número máximo de conexiones:



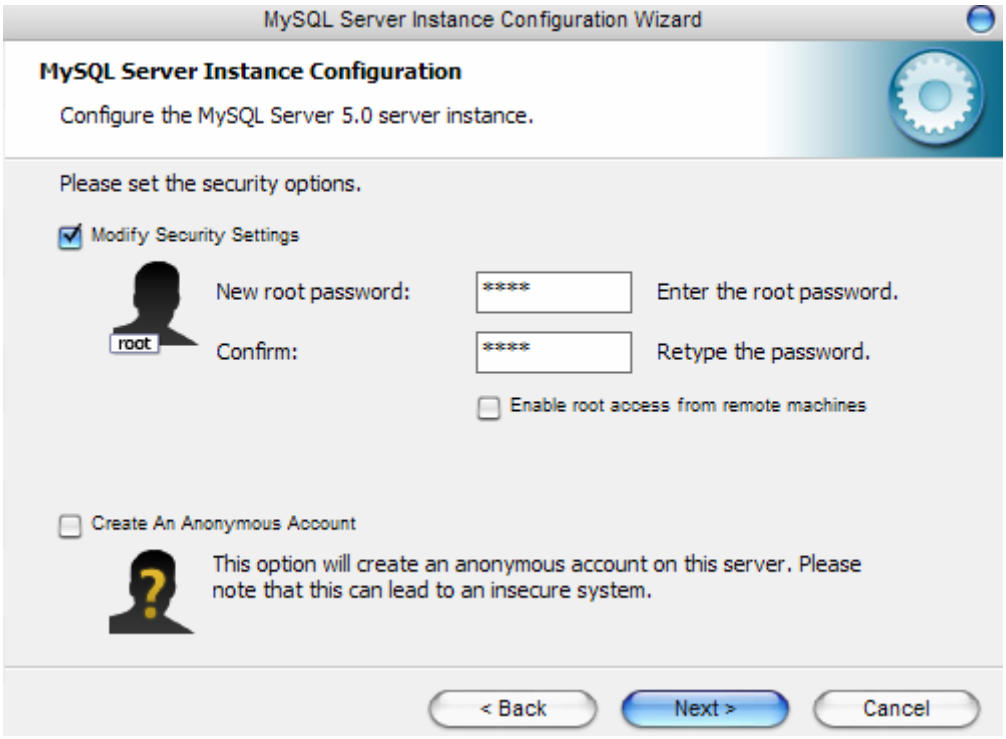
Seleccionar de la misma forma que en la imagen anterior. De esta forma se asume un número reducido de conexiones (20) que serán más que suficientes para comprobar la funcionalidad de nuestra aplicación.

En el siguiente paso de la instalación marcar la opción "Enable TCP/IP Networking" para que nuestros clientes se puedan conectar mediante TCP/IP a nuestro servidor de MySQL. Después como juego de caracteres para la base de datos seleccionar “Estándar Character Set”.
En la siguiente ventana del asistente de instalación elegir esta opción:



De esta forma el servidor de MySQL se instalará como un servicio de Windows y no será necesario iniciarlo manualmente pues se iniciará de forma automática al comenzar el sistema.

El último paso de la instalación es muy importante. Es necesario señalar la opción “Modify Security Settings”. De esta forma podemos seleccionar la contraseña para la base de datos. Se debe introducir como tal “jefe”. Esto es así, porque se utiliza posteriormente en el código jsp para consultar la base de datos. Como no se ha realizado una lectura de parámetros para la contraseña estos son fijos y debe introducirse esta contraseña para el correcto funcionamiento del entorno Web en el momento de comprobar si la contraseña y nombre de usuario están realmente en la base de datos para permitirnos el acceso. A continuación se muestra la captura de pantalla de este último paso en el que debemos fijar la contraseña del servidor de la base de datos.



Tras este último paso, nuestro servidor MySQL estará configurado correctamente. Para restaurar la sencilla base de datos de nuestro sistema recomendamos la aplicación “MySQL Administrator”. Al instalar esta aplicación, que nos permite gestionar nuestras bases de datos de forma visual y sencilla a través de un cómodo interfaz, se nos solicitara un nombre de usuario y contraseña. El nombre de usuario es “root” y la contraseña es la que fijamos anteriormente en nuestro servidor, es decir, “jefe”. Tras proporcionar estos parámetros se inicia la aplicación.

Seleccionamos la opción “Restore” del menú izquierdo. Tras esto pulsamos el botón “Open Backup File” que nos permite seleccionar la base de datos que vamos a recuperar. Elegimos nuestra base de datos que se encuentra en el CD del proyecto, en la carpeta “Archivos de configuración”. El nombre del archivo es “Basededatos.sql”.Tras seleccionarlo, se activa el botón “Star Restore”, lo pulsamos y la base de datos de nuestro proyecto queda restaurada y lista para ser utilizada.

Si se quiere modificar la base de datos, añadiendo nuevos registros o modificar los existentes se puede utilizar “MySQL Quero Browser” que permite una gran rapidez al hacer consultas y modificaciones a la base de datos a través de un sencillo interfaz. Todos los programas nombrados se encuentran en la carpeta del CD del proyecto “Programas”.

4.3 Servidor de aplicaciones Web

El servidor de aplicaciones Web utilizado en el proyecto es el Apache Tomcat. Se encuentra en la carpeta “Programas” del CD del proyecto. Su instalación es sencilla, si bien requiere de una configuración posterior.

Para el instalar el mismo basta lanzar su ejecutable. En el momento que nos solicita el nombre de usuario y la contraseña volvemos a utilizar “jefe” para ambos casos por comodidad. No es necesario modificar la ruta de instalación que proporciona por defecto el instalador.

Una vez instalado el servidor de aplicaciones es necesario comprobar que la instalación ha sido correcta. Para ello debemos situarnos en la carpeta “bin” dentro del directorio de instalación del servidor. Una vez allí ejecutamos “startup.bat”.De esta forma se inicia el servidor (de forma similar para detenerlo deberemos ejecutar “shutdown.bat”). Una vez hecho esto lanzamos nuestro navegador y en la barra de direcciones escribimos: ”`http://localhost:8080/`” , si todo ha salido bien debería aparecer la pagina principal del servidor en nuestro navegador.

A continuación debemos configurar nuestro entorno. Para ello es necesario copiar la carpeta del CD del proyecto “pagina” que se encuentra en el directorio “Aplicación” en la carpeta del servidor “webapps”.De esta forma al iniciar el servidor cargara nuestro entorno para su utilización. Necesitamos también configurar el sistema para incluir el driver MySQL utilizado para la conexión con la base de datos a través de la página JSP. Este driver se encuentra en la carpeta del CD “Archivos de Configuración”, su nombre es “mysql-connector-java-3.1.6-bin.jar”. Debemos copiar este driver en la siguiente carpeta del servidor: “\common\lib”. De esta forma el driver estará disponible para su uso.

Por ultimo es necesario sustituir el archivo de configuración original del servidor por el incluido en el CD , en el que esta incluida la información de nuestro proyecto para su soporte por el servidor. Esto se consigue sustituyendo el archivo “server.xml” de la carpeta “conf” del servidor por el proporcionado en el CD en el directorio “Archivos de configuración” con el mismo nombre.

La instalación ha concluido tras estos pasos. Basta escribir en la barra de nuestro navegador “`http://localhost:8080/pagina`” para que el servidor comience a dar soporte a nuestra aplicación.

5. Experiencias.

Varias de las animaciones se han proporcionado a los alumnos de la asignatura de 'Estructuras de Datos y de la Información' de la I.Superior y de la I.T. de Gestión de la Facultad de Informática de la UCM a través del Campus Virtual. El acceso a las animaciones se realizó conjuntamente con la herramienta interactiva de aprendizaje de estructuras de datos, desarrollada en el proyecto de S.I. 'Visualización y Animación de Estructuras de Datos y Algoritmos' del curso 04/05.

La primera experiencia consistió en proponer una serie de prácticas a los alumnos para facilitarles el aprendizaje de las estructuras de datos. Las prácticas se completaban con un test, con el que los alumnos podían comprobar lo que habían aprendido. Los tests cubrían los tres aspectos fundamentales de las estructuras de datos: comportamiento de la estructura, implementaciones más conocidas, y aplicaciones de la estructura a problemas concretos. En esta experiencia se utilizaron las siguientes animaciones:

- Transformación de una expresión infija en una expresión postfija. Esta animación se proporcionó como ejemplo de aplicación de TAD de las pilas. El test fue respondido por 77 alumnos.
- Recorrido en anchura de un árbol binario. Esta animación se proporcionó como una operación del tipo de datos de los árboles binarios de búsqueda. El test fue respondido por 31 alumnos.
- Inserción de elementos en un árbol roji-negro. Esta animación se proporcionó como ejemplo de árboles equilibrados. Dado que este tema era más específico y no se había explicado en clase la participación fue menor que en los casos anteriores, 18 alumnos.

En la segunda experiencia se formaron dos grupos de alumnos. A los alumnos de uno de los grupos se les proporcionaron las animaciones mientras que a los alumnos del otro grupo se les permitió acceder a los apuntes de clase y a la bibliografía de la asignatura. A todos los alumnos se les proporcionó el mismo test sobre los algoritmos estudiado y se compararon los resultados obtenidos por los dos grupos. Esta experiencia se realizó en un grupo de la I. Superior con 27 alumnos y en un grupo de la I.T. de Gestión con 30 alumnos. La experiencia se realizó durante la hora de clase. Los alumnos que estudiaron los algoritmos utilizando las animaciones lo hicieron en el laboratorio, mientras que los otros permanecieron en la clase. Los algoritmos estudiados fueron: el algoritmo de cálculo de un árbol de recubrimiento mínimo de Prim y de Kruskal.

Los resultados de estas experiencias, así como el desarrollo de las animaciones se han enviado a las III jornadas del Campus Virtual de la UCM [16] y al congreso SIIE'06 [17] para su publicación.

6.- Conclusiones

Una vez terminado el proyecto podemos concluir que los objetivos del mismo se han cumplido:

- Intentar cubrir lo más ampliamente posible el temario de cualquier asignatura de estructuras de datos: se ha hecho un recorrido desde las más simples (pilas y colas) hasta alguna de las más complejas (montículos y grafos), todas ellas acompañadas por ejemplos prácticos y en bastantes casos por algoritmos que se han seguido paso a paso, mostrándole al usuario el efecto de cada instrucción.

- Ser accesible tanto para el estudio de los alumnos como, para poder ser expuesta por profesores y al mismo tiempo ser accesible desde Internet.
  Una de la principales herramientas para conseguir este objetivo ha sido el sistema de navegación de las animaciones, que en el caso que el usuario sea un alumno puede detenerlas a su antojo cuando no entienda algo (o volver para atrás si es que necesita volver a leer algo previo) o si el usuario es un profesor que está mostrándolas en un aula puede pararlas cuando desee para dar una explicación. Al mismo tiempo, debido a que los gráficos que genera Macromedia Flash son vectoriales (mantienen sus proporciones en el caso de que se agranden o se reduzcan) permite que pueda ser maximizado en el ordenador que se ejecuta sin distorsión alguna de la imagen, y por lo tanto en el caso de que deban ser expuestas en un aula de grandes dimensiones el orador podrá estar tranquilo porque todo el mundo podrá ver claramente la animación sin pérdida de calidad. También, debido al gran carácter portable del formato swf se consigue que las animaciones puedan ser accesibles a todo aquel que lo desee a través de la red, y es que pueden reproducirse por medio de cualquier navegador o mediante Flash Player (programa gratuito que se puede encontrar en la Web de Macromedia).

Por otra parte, hay que mencionar que hemos intentado orientar esta memoria de tal modo que sirva para varios fines:

- Ilustrar el desarrollo del proyecto: hemos intentado mostrar qué es lo que hemos hecho, por qué y para qué.

- Servir como guía para aquellos que busquen información a cerca de qué se puede hacer con Macromedia Flash: esta memoria introduce conceptos básico de Macromedia Flash y aporta ideas sobre cómo se consigue generar movimiento en Flash (o cómo

podemos engañar al ojo humano para que sólo él vea movimiento donde lo único que hay es un intercambio constante de imágenes estáticas superpuestas en instantes de tiempo distintos).

- Mostrar a futuros alumnos de la asignatura Sistemas Informáticos interesados en continuar nuestro proyecto en qué consta. Para que puedan hacerse una idea de cómo ampliarlo. Es especialmente importante que quien pretenda continuar este proyecto debe tener presente que las animaciones van dirigidas a personas que en la mayoría de los casos no tienen ningún concepto previo de la asignatura, es decir, deben estar dirigidas en especial hacia los alumnos, por lo tanto, los futuros desarrolladores deben ser capaces de generar los mejores métodos pedagógicos posibles que ayuden a asimilar los conceptos con la mayor prontitud y claridad.

En lo que se refiere a nuestro aprendizaje, no ha sido este un proyecto en que se hayan usado gran cantidad de herramientas, pero sí que se ha pasado mucho tiempo frente a Macromedia Flash (sobre todo), DreamWeaver y libros de estructuras de datos. Gracias ha todo esto hemos obtenido un manejo muy notable de Macromedia Flash y en cuanto a la lectura de los libros nos ha ayudado a ampliar nuestros conceptos de TAD´s y finalmente hemos terminado familiarizándonos con la enseñanza de estos conceptos. Por lo tanto, a parte de las herramientas que hemos aprendido a manejar también hemos aprendido “herramientas” útiles a la hora de transmitir conceptos por esta vía, la de generar animaciones o presentaciones dirigidas a varios tipos de público simultáneamente.

7.- Glosario

A
animación

B

C
css

D
DreamWeaver

E
estructuras de datos

F,G,H,I

J
jsp

K,L

M
Macromedia Flash Player

N,O,P,Q,R

S
Shockwave Flash (swf)

T
tipos abstractos de datos (TAD), tomcat

U,V

W
Web

X,Y,Z

8.- Bibliografía

1. Macromedia, Macromedia Flash. Getting Started with Flash, Macromedia, 154 págs.

2. Macromedia, Macromedia Flash. Using Flash, Macromedia, 678 págs.

3. Macromedia, Macromedia Flash. Flash Tutorials, Macromedia, 290 págs.

4. Mark Allen Weiss, Estructuras de datos en Java, 1ª edición, Addison Wesley, 2000, 776 págs.

5. Narciso Martí-Yolanda Ortega-José Alberto Verdejo, Estructuras de datos y métodos algorítmicos, 1ª edición, Prentice Hall, 2004, 531 págs.

6. Xavier Franch, Estructuras de datos. Especificación, diseño e implementación, 3ª edición, Ediciones UPC, 1999, 423 págs.

7. Simon Brown..., Professional JSP, 2nd edition, Wrox Press, 2002, 1195 págs.

8. Bruce W. Perry**,** Java Servlet and JSP cookbook**,** O'Reilly, 2004, 723 págs.

10. Dan Shafer**,** HTML Utopia: designing without tables using CSS Sitepoint, 2004, 489 págs.

11. Richard Mansfield, CSS Web Design For Dummies, Wiley Publishing, Inc, 2005, 385 págs.

12. MySQL, MySQL 5.0 Reference Manual, MySQL, 1997-2006 (versión online: `http://dev.mysql.com/doc/refman/5.0/en/`)

13. MySQL, MySQL Query Browser Manual, MySQL 1997-2006 (versión online: `http://dev.mysql.com/doc/query-browser/en/index.html`)

14. MySQL, MySQL 5.0 Administrator Manual, MySQL, 1997-2006 (versión online: `http://dev.mysql.com/doc/administrator/en/index.html`)

15. The Apache Jakarta Tomcat 5 Servlet/JSP Container Documentation (versión online: `http://tomcat.apache.org/tomcat-5.0-doc/index.html`)

9.- Autorización del proyecto

Los autores de este proyecto, Roberto Carrasco Sánchez, José Javier Crespo Ábalos y Antonio José López Mancheño, de la asignatura Sistemas Informáticos, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, tanto la propia memoria, como el código y/o la documentación, siempre que seamos mencionados expresamente como los autores.

Roberto Carrasco Sánchez

José Javier Crespo Ábalos

Antonio José López Mancheño